6.858 Fall 2012 Lab 7: Final project

Piazza idea discussions due:	Monday, October 29, 2012
Proposals due:	Monday, November 5, 2012
Presentations due:	Wednesday, December 12, 2012 (in class)
Code and write-up due:	Friday, December 14, 2012 (5:00pm)

Introduction

In this lab, you will work on a final project of your own choice. Unlike in previous labs, you may work in groups of 3-4 for the final project. You will be required to turn in both your code and a short write-up describing the design and implementation of your project, and to make a short in-class presentation about your work.

The primary requirement is that your project be something interesting. Your project should also have something to do with security, but that's relatively easy, and it's much more important for your project to be interesting. lol

Below are some ideas for final projects that you might use as inspiration, including some of the projects from past years. We encourage you to come up with your own ideas for what you would like to work on; there's no need to restrict yourself to this list.

- Build a static analysis tool to find bugs in real web applications. It would be great to have a static analysis tool for Python programs, even if it's not perfectly sound, much like the PHP static analysis tool we discussed in lecture. In addition to the standard XSS bugs, can you find bugs where application developers forget to perform permission checks, or inadvertently leak sensitive data? Such a tool might also be useful to find non-security bugs in Python programs. One existing tool that's quite limited is PyLint.
- Find bugs in real C code. Take a look at some recent research papers on finding real bugs in the Linux kernel and other C-based programs, involving integer errors or undefined behavior. We can give you access to our source code to these tools; you can apply them to existing software to see what bugs you can find, and also think about ways to extend the tools to make them more accurate, to make them find other kinds of bugs, etc.
- Extend program analysis tools used by Linux kernel developers, such as sparse and smatch, to catch different kinds of bugs in the kernel.
- Use the KLEE symbolic execution system to build an analysis tool that finds interesting bugs in C programs.
- Explore the extent to which covert channels / side channels matter, e.g. in shared VMs like EC2, vs. shared OS, vs. other environments. See this paper for some background information.
- Add Capsicum support to Linux.
- Port an interesting application to use Capsicum (on FreeBSD).
- Analyze the security of MIT's single-signon system, Touchstone / Shibboleth. ← I like that one
- Find an interesting application / use case for Webathena (site, code), or extend it to support non-DES enctypes (AES, etc).
- Implement Kerberos for Android. We can put you in touch with some folks at the Kerberos

Consortium that are working on this. It would be interesting to implement a Kerberos ticket caching service, accessed via Android intents, so that an application can use Kerberos tickets for a specific service without being able to steal the user's entire TGS ticket.

- Implement progressive authentication: instead of requiring the user to log in to do anything, require different levels of credentials for specific tasks. For example, on Athena, perhaps running a web browser should not require any credentials, but accessing your personal files (or your browser accessing your `google.com` cookie that gives access to gmail) should require your Kerberos password, etc. The same would apply on an Android phone: checking the weather or the map should not require any credentials, but accessing the mail app should prompt for a PIN, swipe pattern, or password. A paper from Microsoft Research might give you some things to think about, although it may not be worthwhile to implement all of the environment monitoring done in that work.
- Examine the security of Android applications. Look at some previous studies for inspiration (one, two).
- Improve sandboxing for Android applications. Is there something that you could do to prevent a malicious application from exploiting kernel bugs? Consider recent improvements to seccomp, using Linux KVM, or using Native Client.
- Audit interactions between Android applications, perhaps by tracing all intents sent via the reference monitor. When something goes wrong, can your system tell the user why an application might be broken?
- Implement an environmental key generation system.
- Build a password manager for Android applications, perhaps by implementing a virtual keyboard that can automatically enter passwords into an application. The virtual keyboard can know exactly what application it's entering passwords into, which can guard against phishing-style attacks. LastPass does
- Provide more fine-grained network access control in Android, to protect an internal corporate network from possibly-malicious applications on a phone. would be cool but hard
- Allow users to control application permissions in Android. For example, a user may want to install some application that requires access to the current GPS location, but the user doesn't actually want to give the app this permission. Would it suffice to simply remove the permission from the manifest, or is it necessary to provide dummy services that give back fake data?
- Implement more efficient sandboxing support for Python, so that it's possible to import an untrusted Python module without giving it full access to your system or your application, and without having to run the entire application in a separate PyPy sandbox. Python already provides some basic memory safety guarantees (although you might need to implement some mechanism to restrict introspection).
- Speed up PyPy's sandbox mode. lol
- Implement a more fully-featured PyPy sandbox. Can you run all of zoobar in the sandbox, to avoid any need for uid-based privilege isolation? what's stopping you? - do you read diff Sandboxes?
- Implement a buffer overflow protection scheme, perhaps similar to Baggy bounds checking, building on top of Clang and LLVM.
- Use DynamoRIO's binary instrumentation to implement some cool security mechanism for unmodified binaries. For example, you could implement a binary-level taint tracking system, similar to Resin, that prevents secret files from being sent out over the network. You can look at a previous lab we used to have involving DynamoRIO from a few years ago here.
- Use Resin's taint tracking for Python to enforce some interesting properties in zoobar. research Resin →
- Find an interesting use for trusted hardware, and figure out how to expose trusted hardware safely to applications. Linux should already have a basic device driver for a TPM.
- Write a tool to help privilege-separating Python applications.

- Evaluate how hard it is to privilege-separate a real application (pick any application and try to privilege-separate it yourself; there should be many examples of Django-based Python apps).
- Implement more flexible protection mechanisms for Linux (so that any user can create additional protection domains -- sub-users -- to run code with less privileges, without having to be root). You can build upon a class project from a previous year, UserFS.
- Based on Google's caja library, sandbox existing Javascript mashups/applets (what to do about existing uses of globals in the Javascript environment?)
- Write a browser plugin to prevent cross-site scripting attacks when both the server and the client are following some rules (e.g. explicitly annotating privileged JS code). Bonus points for allowing untrusted JS code using something like caja!
- Improve the security of HTTPS in web browsers in the face of possibly compromised CAs. For example, define a new URL syntax that includes the server's certificate public key in the URL itself, so that one site can unambiguously include a link to another site without relying on CAs. Or, include the CA name in the URL, so that another CA cannot subvert security. See SSL observatory, perspectives, and CertPatrol.
- Implement an encrypted file system with plausible deniability (i.e. where there can be multiple encrypted file system images within a single FS, and without the right password, you don't know if unused blocks are free or part of another encrypted FS you don't have the password for.) See paper on deniable file systems and TrueCrypt.
- Auditing support for web applications. For instance, suppose someone broke into your blog or forum, added a user account for themselves, changed permissions, and posted garbage messages. How can you track down all of the changes made by the attacker? Could be done with the help of a language runtime, such as Resin.
- Examine the security of Google's Chromebook laptops. We can loan out some CR-48 laptops to interested students.
- Integrate zoobar with single-signon protocols like OAuth or OpenID.
- Figure out how to integrate password-authenticated key exchange protocols into a web browser. Note that such protocols could be used to both authenticate the user to the server (thus avoiding phishing attacks), and to authenticate the server to the user (thus avoiding compromised CA attacks). For references, look up SRP and PAKE.
- The Tor Project has some ideas for possible Tor-related projects here.
- Analyze the Bitcoin transaction graph. How hard is it to anonymize Bitcoin exchanges? Here's one recent paper analyzing the Bitcoin data set, which might give you ideas for other things to try.

We encourage final projects that combine some ideas you have learned in 6.858 with other classes or projects you are already working on. For example, implementing some aspects of a capability design from Capsicum in the JOS kernel from 6.828 might make a good final project that you could use in both classes. Extending some system you are already working on to add better security mechanisms would also be a good candidate project.

Several final projects from last year's class ended up being subsequently published as research papers (e.g., UserFS, BStore, and LXFI). If this sounds interesting to you, try to pick an ambitious class project that you might want to continue working on afterwards!

There are four concrete steps to the final project, as follows:

Form a group. Decide on the project you would like to work on, and post short summary

of your idea (one to two paragraphs) on Piazza. Discuss ideas with others in comments on their Piazza posting. Use these postings to help find other students interested in similar ideas for forming a group. Course staff will provide feedback on project ideas on Piazza; if you'd like more detailed feedback, come chat with us in person.

Project proposal. Discuss your proposed idea with course staff over the next week, before the proposal deadline, to flesh out the exact problem you will be addressing, how you will go about doing it, and what tools you might need in the process. By the proposal deadline, you must submit a one-to-two-page proposal describing: your **group members** list, **the problem** you want to address, **how you plan to address it**, and what are you proposing to **specifically design and implement**.

Submit your proposal to the submission web site.

Project presentation. Prepare a short in-class presentation about the work that you have done for your final project. We will provide a projector that you can use to demonstrate your project.

Write-up and code. Write a document describing the design and implementation of your project, and turn it in along with your project's code by the final deadline.

Ideas for Lab 7

10/29

Jhang + I talked earlier
- forgot already

Something around passwords
Best practices manual
include typing speed in password

Single Sign On?

GRL Password Gid

Phishing filter?

SSL still same - good extension...

---

look into touchstone
? library for (PHP) OAuth

(2)

Progressive authentication seems cool
but how to actually do?

Chromebook

Bitcoin

(I'm bad at thinking of ideas...)
esp in security
don't know everything at there

But need to also join groups
We don't each have to contribute idea

Pharos security

NFC

RFID

Hubway

(3)

HW OTP token

PAM - pluggable authentication module

Zeo sleep manager

Progressive authentication for android

Something w/ NaCl

Hack grading system

Type in diff password
like backwards

Enter on keypad ↓ ↑ → ←
had from watching
grid randomly gen
transmitted to ya

(4)

; give starting point

longer password = less secure

6-10 characters

Swap up-down
left-right

English words bad
-quick analysis

For each letter
was it up or down
was it left + right
mean down set
More letters → doesn't help

; quads too ;

(5)

Sandra show known password

We actually txn it

Other possibe variations

Write up

Would require storing plain text...

unanswered question 1 views

Final Project: Novel Password Systems Where Enter Derivation of Password instead of Actual Password

We propose exploring systems where the user enters a representation of their password, instead of their actual password.

A basic example of this is ING Direct's PIN pad where the user enters the letters corresponding to their PIN instead of the PIN itself. The mapping between numbers and letters is randomly generated every log in. Obviously if the attacker had the mapping then the game is over - but this requires more than a simple keylogger.

For example, what if every time you logged in, you were given a grid like system of 26x26 letters - each letter only once per row and column (a Latin Square - like Sudoku). You are given a start point. Then you trace out your password alternating between rows and columns. You enter the directions (up, down, left, right) that you follow. The system can easily verify your password. However an attacker with knowledge of the grid and directions can not easily determine your password, especially if it is not a dictionary word. Now there are quite a few limitations - like knowing a lot of grids, and storing the password in plain text on the server, but like the vast array of proposed passwords ideas - it is locally optimal for a specific case. (The original inspiration is from https://www.grc.com/otg/operation.htm)

Still we think ideas like these would be interesting to explore even if we don't come up with a revolutionary new password scheme.

-theplaz and jwang7

#final_project_idea

edit save to favorites 0 Just now by Michael Plasmeier 1 edit

the students' answer, *where students collectively construct a single answer*

Click to start off the students' answer

followup discussions, *for lingering questions and comments*

Proposal

11/5

TA suggested picture password
How much do we want to consider that?

That is not when you enter a representation of password

Generalize

11/5

Novel Password Systems Where Enter Derivation of Password instead of Actual Password

6.858 Proposal

Michael Plasmeier <theplaz>
Jonathan Wang <jwang7>
Miguel Flores <mflores>

We propose exploring systems where the user enters a representation of their password, instead of their actual password.

The Problem

The problem with many password systems is that users must type their entire, full password each time they log on. This makes the password vulnerable to key logging and interception during transmission.

How we plan to address it

We seek to explore systems in which the user does not enter their direct password, but a derivation of the password **which changes on each log in**. The user proves that he or she knows the password without subsequently ever providing the password itself.

ING Password Keyboard

A simple example is ING Direct's PIN pad. Under ING's system, the user enters the letters corresponding to their PIN instead of the PIN itself. The mapping between numbers and letters is randomly generated on every log in. This method does not survive an attack where the attacker has access to the mapping, but it does prevent simple keylogging.

Figure 1 ING's Pin Pad. The user enters the letters corresponding to their PIN in the box.

Windows 8 Picture Passwords

Our proposal does not cover systems such as Windows 8's new "Picture Passwords" feature. With Picture Passwords, the user's password is a series of user selected vectors which a user draws on top of a user specified picture. However, the vector must be repeated exactly the same each time. We propose to concentrate on systems where the user enters a different derivation of their password each time.

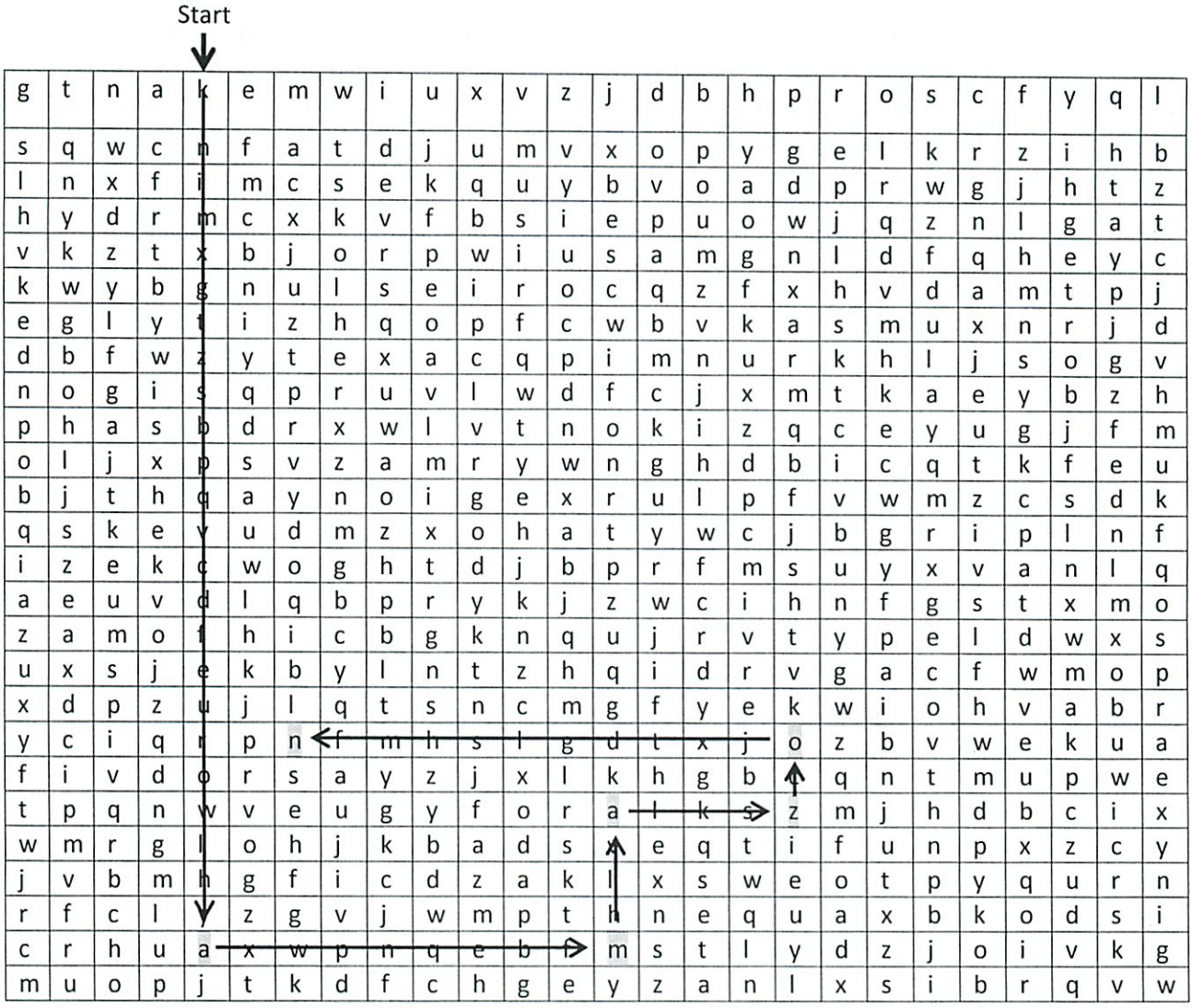
Trace the Grid

Under this system the user chooses a password from a lower case Latin alphabet. However, instead of entering the password each time the user logs in, the user is presented with a randomized grid of 26x26 letters with each letter only once per row and column (this is called a Latin Square. Sudoku is the most famous example of a Latin Square). The user then enters a derivation, called a *trace* of their password which is transmitted back to the server.

The server first randomly generates a 26x26 Latin square and a start row or column. These are transmitted to the user. The user then visually traces out his or her password on the grid, alternating between rows and columns. For example, the user would locate the first letter of their password on the start row or column. The user would then look for the next letter of his or her password in either the column (if the start was a row) or row (if the start was a column) that contained the user's first character. The user would then continue alternating for the length of their password.

The user enters the directions (up, down, left, right) that they follow as they trace out their password. The user should include the direction from the start marker. This combination of directions that the user inputs is referred to as a *trace*. The trace is then sent from the client to the server, and the server can easily verify that a trace corresponds to the correct password.

Example: entering the password Amazon with the 5th column as the start row/column. The grid as well as the start row/column are randomly generated be the server for each log in.



g	t	n	a	k	e	m	w	i	u	x	v	z	j	d	b	h	p	r	o	s	c	f	y	q	l
s	q	w	c	n	f	a	t	d	j	u	m	v	x	o	p	y	g	e	l	k	r	z	i	h	b
l	n	x	f	i	m	c	s	e	k	q	u	y	b	v	o	a	d	p	r	w	g	j	h	t	z
h	y	d	r	m	c	x	k	v	f	b	s	i	e	p	u	o	w	j	q	z	n	l	g	a	t
v	k	z	t	x	b	j	o	r	p	w	i	u	s	a	m	g	n	l	d	f	q	h	e	y	c
k	w	y	b	g	n	u	l	s	e	i	r	o	c	q	z	f	x	h	v	d	a	m	t	p	j
e	g	l	y	t	i	z	h	q	o	p	f	c	w	b	v	k	a	s	m	u	x	n	r	j	d
d	b	f	w	z	y	t	e	x	a	c	q	p	i	m	n	u	r	k	h	l	j	s	o	g	v
n	o	g	i	s	q	p	r	u	v	l	w	d	f	c	j	x	m	t	k	a	e	y	b	z	h
p	h	a	s	b	d	r	x	w	l	v	t	n	o	k	i	z	q	c	e	y	u	g	j	f	m
o	l	j	x	p	s	v	z	a	m	r	y	w	n	g	h	d	b	i	c	q	t	k	f	e	u
b	j	t	h	q	a	y	n	o	i	g	e	x	r	u	l	p	f	v	w	m	z	c	s	d	k
q	s	k	e	v	u	d	m	z	x	o	h	a	t	y	w	c	j	b	g	r	i	p	l	n	f
i	z	e	k	c	w	o	g	h	t	d	j	b	p	r	f	m	s	u	y	x	v	a	n	l	q
a	e	u	v	d	l	q	b	p	r	y	k	j	z	w	c	i	h	n	f	g	s	t	x	m	o
z	a	m	o	f	h	i	c	b	g	k	n	q	u	j	r	v	t	y	p	e	l	d	w	x	s
u	x	s	j	e	k	b	y	l	n	t	z	h	q	i	d	r	v	g	a	c	f	w	m	o	p
x	d	p	z	u	j	l	q	t	s	n	c	m	g	f	y	e	k	w	i	o	h	v	a	b	r
y	c	i	q	r	p	n	f	m	h	s	l	g	d	t	x	j	o	z	b	v	w	e	k	u	a
f	i	v	d	o	r	s	a	y	z	j	x	l	k	h	g	b	q	n	t	m	u	p	w	e
t	p	q	n	w	v	e	u	g	y	f	o	r	a	l	k	s	z	m	j	h	d	b	c	i	x
w	m	r	g	l	o	h	j	k	b	a	d	s	x	e	q	t	i	f	u	n	p	x	z	c	y
j	v	b	m	h	g	f	i	c	d	z	a	k	l	x	s	w	e	o	t	p	y	q	u	r	n
r	f	c	l	y	z	g	v	j	w	m	p	t	h	n	e	q	u	a	x	b	k	o	d	s	i
c	r	h	u	a	x	w	p	n	q	e	b	f	m	s	t	l	y	d	z	j	o	i	v	k	g
m	u	o	p	j	t	k	d	f	c	h	g	e	y	z	a	n	l	x	s	i	b	r	q	v	w

The resulting trace would be: Down, Left, Up, Right, Up, Left.

Figure 2 A trace of the password "amazon"

An attacker with knowledge of the grid, including the start location and the *trace,* cannot easily determine your password.

Some limitations of the system include:

- an attacker can eventually figure out the password by observing enough grids and traces
- passwords are stored in plain text on the server
- a user moving the mouse pointer or finger to manually trace out the password on the grid could be observed by an "over the shoulder" adversary

However, this scheme allows for the trace to be observed.

What we plan to build

We plan to build implement of at least one of the systems we describe. We will build a library for Python Flask which would allow developers to easily adopt this authentication system. The library would insert the grid of letters into the login page that it serves, and it would process the user input to verify that it correctly traces out the password in the grid. The system will also impose restrictions on the passwords that a user can have in order to lower the chance of randomly guessing a correct trace.

Building the system would help us better understand the amount of effort needed to implement the system. We will also run usability testing to evaluate the system which we implement. This would help us comment on how easy it is for users to adopt the new system.

6.858 11/25

Mfg

Final Project

Pres 12/12

Code + writeup 12/14 @ 5PM

Design + implementation

3 systems

1. Original off the grid
2. Our last proposal
3. Make a new one

Security

Prob of guessing

- brute force
- observe transmission
- over the shoulder
- browser

- once vs over time
- word vs random

(2)

- Visibility
  - UI class

- Other factors in that paper we read
  - ease of implementation
  - 20-30 factors

- Write up exact rules
  - What type of passwords
  - # of tries before lockout
  - How PW stored on backend

- ≥10 pgs

Code is pretty much done
he needs to populate letters uniquely

Smaller square
- not 26 x 26

(3)

Your alt theme

Can't have consec letters in Latin sq

aa

14x14 reduced version

told ya guess from looking at repeats of i

---

all systems, 1 type analysis

- Twang - code

  Plaz - write up rules
  - visibility
  - other factors

  Miguel - Prob of guessing

(9)

Code

Virtual env — so when you run it works it runs

Flask

SQL alchemy

JQuery

Draft done ~~12/10~~ 12/2

possibly 12/10

On Skydive

Michael E Plasmeier

From: Nickolai Zeldovich <nickolai@csail.mit.edu>
Sent: Monday, November 12, 2012 11:39 AM
To: Michael E Plasmeier; Jonathan Wang; Miguel A Flores
Cc: 6.858-staff@amsterdam.lcs.mit.edu
Subject: 6.858 project proposal

Hi,

Below are some comments on your 6.858 final project proposal from the course staff. Feel free to send us email if anything is unclear.

Nickolai, David, Taesoo, and Frank.

Novel Password Systems Where Enter Derivation of Password instead of Actual Password

Cute! I'd encourage you to think about how secure / usable / deployable your scheme is. If any of you have taken Rob Miller's UI class, perhaps you can actually say something more intelligent than me about usability.
At the very least, the "Quest for password replacement" paper should give you some things to consider, and pointers to prior schemes that work in a similar way, so that perhaps you can improve on them.

WORK IN PROGRESS

Off The Grid

WORK IN PROGRESS

A paper-based system for encrypting domain names into secure passwords.

Sample Grid Only – See page links belowfor usage instructions and personal grid creation

	+	|	?	2	~	[	{	0	1	2	3	4	5	6	7	8	9	-	.	&	3	6	&	@	?	|	
)	w	z	u	P	F	l	G	r	E	J	v	C	Y	I	n	K	q	o	D	t	X	b	s	M	H	A	0
'	R	P	X	w	O	E	u	G	n	y	J	f	M	D	I	s	h	K	v	b	C	Z	l	A	Q	t	9
(	X	g	T	z	p	s	L	a	I	B	u	d	w	J	c	f	M	V	Q	h	n	y	K	e	r	o	6
1	M	a	l	D	G	w	O	y	H	I	q	u	V	r	s	J	n	c	k	X	B	E	f	t	Z	p	#
0	c	L	o	g	B	R	D	q	W	U	f	P	e	K	m	t	A	z	N	V	I	s	y	x	J	H	[
9	u	m	n	I	V	K	f	H	c	s	E	Z	Q	g	T	x	B	Y	a	D	R	p	W	O	l	J	)
4	B	h	V	U	m	i	e	P	R	O	n	Q	k	y	j	Z	D	w	F	s	l	A	t	g	x	C	:
}	Y	N	i	c	L	G	K	b	J	h	D	r	S	A	E	o	z	T	p	m	Q	V	x	F	u	w	9
>	f	S	H	Y	c	P	m	E	D	k	Z	I	X	w	B	Q	R	n	t	u	G	o	v	J	A	L	2
6	J	k	a	T	n	b	H	i	o	X	r	S	G	f	l	D	p	M	u	E	V	w	Z	Q	c	y	;
1	e	b	F	x	Y	v	w	m	T	g	K	A	p	u	Z	N	l	q	C	O	j	D	H	i	s	R	@
)	q	j	y	H	X	a	P	k	g	N	w	t	Z	V	r	i	U	D	O	l	M	f	C	b	E	S	5
\	h	Q	W	k	R	Z	j	D	M	C	x	e	A	B	V	g	S	I	Y	p	U	n	o	l	T	F	1
8	G	Y	m	S	h	x	n	J	v	D	C	B	t	e	K	L	W	r	i	a	O	Q	p	Z	F	u	/
~	L	X	g	A	e	H	b	c	S	w	T	v	O	q	D	p	i	F	Z	K	y	U	j	r	m	N	3
7	o	w	b	N	Q	m	V	x	U	a	S	Y	c	L	g	e	f	P	H	J	k	T	R	D	i	Z	\
0	n	f	D	l	Z	t	X	U	B	r	y	O	J	P	q	h	g	E	M	I	s	c	A	v	W	k	'
3	a	R	p	f	s	C	Q	t	x	Z	I	j	L	o	h	B	Y	u	w	n	D	g	E	K	v	m	*
&	Z	I	r	m	a	Q	t	W	Y	l	b	n	H	C	X	v	j	S	g	f	e	K	u	P	o	d	9
'	i	u	s	E	T	J	y	o	Q	V	a	h	d	M	f	c	k	L	x	z	p	R	N	W	g	B	3
<	k	V	C	j	u	f	I	L	z	P	O	M	r	n	a	W	X	b	E	G	T	h	D	s	Y	Q	7
2	v	E	q	O	j	D	C	z	k	M	P	G	f	S	U	r	t	h	L	W	a	I	b	Y	n	X	\
5	D	O	k	Q	w	Y	r	F	A	t	h	L	N	x	P	M	V	g	S	C	z	j	i	U	b	e	/
!	S	t	E	v	D	u	A	N	p	q	G	w	B	Z	O	y	C	X	j	r	F	l	m	H	k	i	4
'	T	D	J	b	k	N	z	s	f	E	l	X	i	h	y	U	o	a	r	Q	w	m	G	c	P	v	8
0	p	c	z	R	i	o	s	V	l	f	M	K	u	t	W	a	E	J	b	Y	h	X	Q	N	d	G	{
	1	3	2	/	9	4	6	(	^	+	'	~	@	%	(	:	}	9	9	8	<	]	9	5	0	7	

Re-Generate Encryption Grid

- Watch it Work
- Get it Done

The grid above contains a highly random arrangement of characters, but with **very special properties**. It can be used to encrypt website domain names into secure passwords.

Why is this useful?

This "**Off The Grid**" technology is the *only known system* to provide secure encryption using nothing but a specially designed piece of paper.

Although this system initially uses software to design and print the grid, *no technology of any kind* is used to perform the encryption. Every other strong and modern encryption technology relies upon software running in your computer, your

browser, your phone, or some other device to encrypt, decrypt, or store your passwords. And as everyone knows, *any software-based system can be compromised by malware* . . . and sooner or later, most are.

We designed this "Off The Grid" system to provide Internet users a *uniquely and provably secure* alternative to simply **hoping** that malicious software hasn't, or won't, infiltrate their personal password keeper, online password storage system, on-the-fly password generator, computer-hosted password storage vault, or whatever convenience technology is being employed to generate and/or store the passwords used to authenticate the user's identity to remote Internet websites.

"Off The Grid" converts any website's name into a secure password that you never need to write down, store, or remember because you can easily re-create the same secure password from the same website name the next time, and every time, you need it.

Websites are routinely compromised with their users' logon identity (eMail address and password) stolen. So reusing the same password on separate websites creates a tremendous risk because bad guys could obtain your eMail address and password from one site, then logon as you somewhere else with your reused password.

The "Off The Grid" system securely and uniquely encrypts each website's domain name into your personal password for that one site, so it automatically creates a different secure password for each website and reuse never occurs.

Once you know how to use your own unique personal grid, you will be able to easily create secure passwords that are "Off The Grid" so that no possible compromise of your computers, phones, "the cloud", or any other devices can compromise your security.

Even though **we** can no longer live "off the grid" . . . at least our **passwords** can!

The pages linked below thoroughly explain the system's security goals and operation, and allow you to create, print and begin using your own customized OTG grid:

Off The Grid Resource Pages:

1	"Off The Grid" Introduction	**6**	Frequently Asked Questions
2	Security Goals and Design	**7**	Technical Details and Docs
3	How to use the OTG system	**8**	Security & Attack Analysis
4	Enhanced Security Options	**9**	Latin Squares Workbench
5	Create/Print Your Own Grid	**10**	"Off The Grid" Feedback

GRC's Ultra-High Entropy Pseudo-Random Number Generator

Jump To Top

WORK IN PROGRESS

Off The Grid

WORK IN PROGRESS

Security Features, Goals & Design

AN UNFORTUNATE TRUTH: The ongoing problem is website security breaches conclusively demonstrates that sites which require us to login and authenticate our identities **cannot be trusted** to keep our passwords secret. This makes password reuse at multiple sites extremely unsafe. If one site leaks our name, eMail address, and password to a hacker they attempt to reuse that authentication to impersonate us on other sites.

Or, as the infamous Lulz Security hacking group, "LulzSec", tweeted at 10:34am, Friday, June 24th, 2011 . . . phrasing it somewhat less delicately:

> **@LulzSec:** Reusing passwords is kind of like owning multiple houses and using the same key for each one. Don't expect people not to steal your shit.
> *Fri 24 Jun 10:34 via web (received via TweetDeck)*

Until an industry-wide, coherent identity authentication solution is established, the responsibility for creating a potentially limitless number of separate website "identities" rests with each individual.

WHAT WE NEED:
We need to somehow arrange to use a **different, strong, secure and complex password** for each site that requires us to invent an identity so that we can reauthenticate our identity upon our subsequent return.

Security conscious users know that passwords need to be complex and long to be safe. And GRC's Password Haystacks password padding approach offers one solution in this battle to construct secure **and** memorable passwords. But the trouble is, **we need to create a potentially unlimited number of unique passwords**. It's one thing to create and memorize and/or record a strong **and unique** password where we have to, for our most important sites, such as banking and eCommerce. But today we're asked to create passwords even for "throwaway" sites we visit once and may never return to, just to post some feedback in a forum or blog. And if we do return, what was that password we created the last time?

The problem has been so intractable and pervasive that many "high-tech" and highly useful solutions, such as LastPass, KeePass, and SuperGenPass have been created to lift some of the password management burden from overwhelmed users. But all of these solutions also have liabilities. In mid 2011, LastPass users had a scare when it was revealed that some of its users' database may have escaped LastPass' control. It's convenient to have all of our authentication information stored "in the cloud" . . . but only so long as it is **never** stolen. And it has been demonstrated that SuperGenPass users may be exposing their critical master password to malicious websites.

We have seen over and over that anything which

relies upon technology can be compromised.

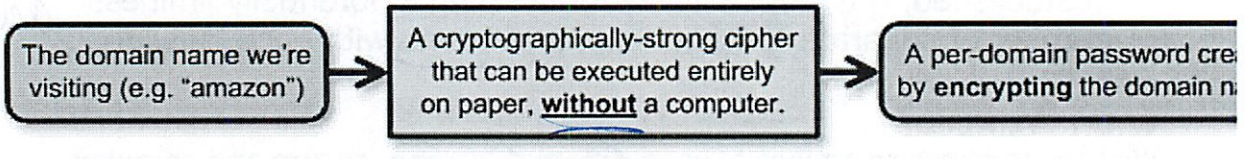
The other concern with cloud-based storage is availability. It's convenient . . . as long as the service is available. Also in mid 2011, the United States FBI (Federal Bureau of Investigation) confiscated three "racks" worth of web servers, reportedly because they could not be bothered to determine which single server among them was believed to be violating the law. In the process, several score of unrelated web sites disappeared from the Internet. We would not be happy if the cloud-based password manager we depend upon was among them.

For these reasons, among others, many users refuse to centralize their password management.

Faced with these many, and growing, problems . . . a new solution was needed.

Immediately after finishing the work on the Password Haystacks password padding approach, I wanted to look in a different direction. My idea was to see whether I could design a secure cryptographic "paper cipher" requiring, for its use, **no** instrumentality, **no** technology, **no** computers, **no** software, **no** wires — **only a simple piece of paper of some kind.** A computer would certainly be required to design and print any instance of the Cipher. But once that was done, no computer would be required to **use** it.

This is the core of the idea I started with:



The idea of using a computer to encrypt a domain name to create a per-domain password is not new. That's the idea underlying SuperGenPass and others. Its obvious benefit is that instead of needing to record, store, or memorize random passwords that we invent per domain — with the potential problems that invites — we employ an **algorithm** of some sort to create — and recreate in the future — domain-name-based passwords. Then we don't need to record, store, or memorize them because we can simply recreate the same password from the same domain name any time it's needed. It's a great idea with no obvious drawbacks . . . except that all available solutions are "online" (in one sense or another) and suffer from the potential of worrisome privacy and security breach problems.

What would such a system look like? What would its requirements be?

Here were my requirements:

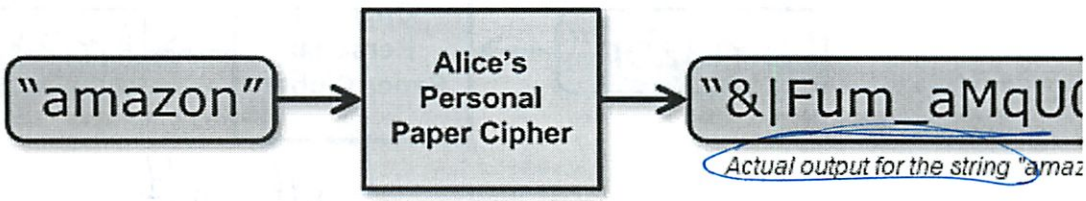
- **Easy to use**
  I wasn't doing this as an intellectual curiosity. My absolute goal was to create a solution that **anyone** interested in creating secure passwords could learn to use quickly, and would then find to be useful and even fun to use.

- **Secure**

The system's security was, of course, as crucial as ease of use. This new system needed to be at least as secure as anything and everything it was replacing so that users could be confident that they were generating passwords that could not be "reverse engineered" to learn anything that might render the system less than absolutely secure. And by assisting its users to create much more secure passwords than they were normally likely to, it would be proactively increasing their security.

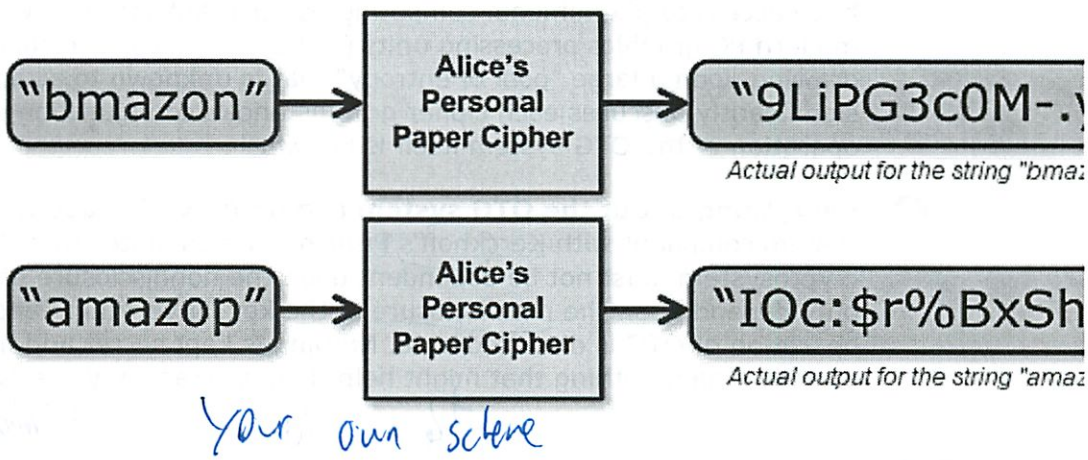
What *EXACTLY* do we mean by "high security"?

1. **The encrypted password output must be long enough to thwart brute forc attacks.** The Off The Grid (OTG) expands every case-insensitive input character (a pair of unpredictable characters. The first six (input) characters of a domain nar therefore expanded into 12 output characters:



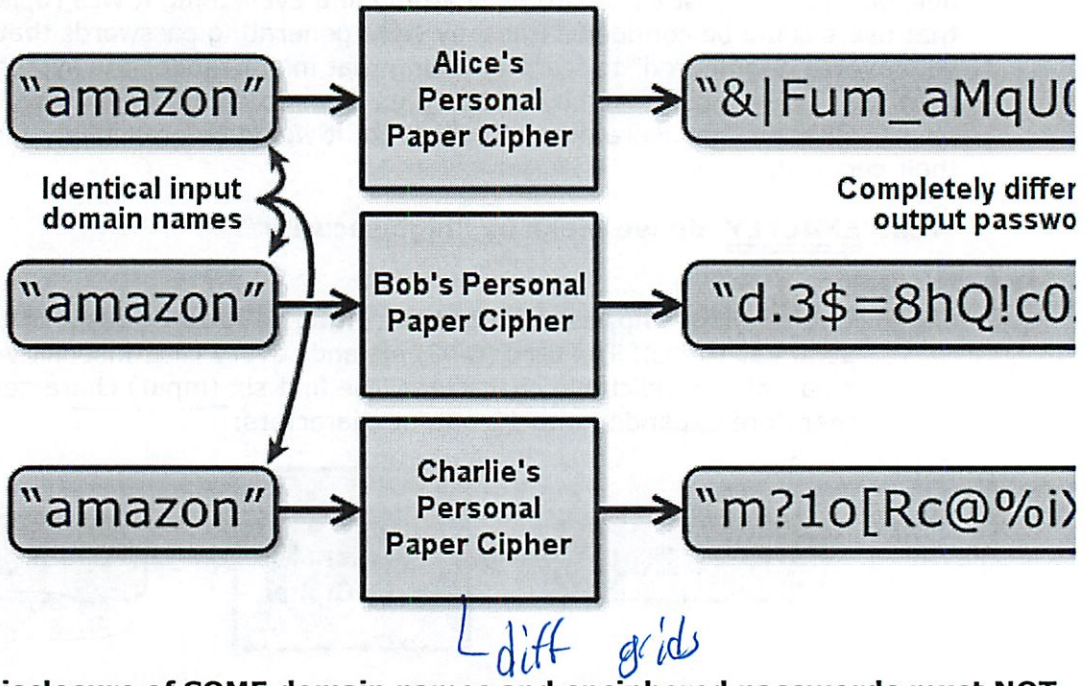
A very short domain name such as "grc" can be enciphered as "grc.co", using the dot (.) and some of the top level domain name to obtain six input characters. And see, even shorter domain names can also be expanded into 12 enciphered charac

2. **The enciphered output depends upon ALL input characters.** This is an impor property for high security. A minimal change to the input must result in a maximu unpredictable change in the enciphered output. In the two examples below – whic generated by the OTG system – only the first and the last character was changed domain name "amazon":



Your own scheme

3. **Every user's enciphered output is completely different from that of every** When any user enciphers a domain name — even the same domain name — usin own OTG cipher, they will obtain a completely unique result, different from any o user:



4. **Disclosure of SOME domain names and enciphered passwords must NOT compromise the security of ANY other passwords.** Even if an attacker knew were generating your passwords with the OTG system — and that is not obvious f output — and if an attacker were to somehow acquire some of your passwords ge by the OTG system, it is imperative that so little about your OTG cipher could be determined that **none** of your other passwords would be weakened. As we will see OTG system achieves this by embedding an extremely **large** amount of "entropy" (randomly determined data) into each instance of a user's personal, custom ciphe

5. **Resistance to "computational" attack.** Today's computer hobbyists (and attack have access to phenomenal computing power thanks to the awesome power built modern PC graphics processing units (GPUs). OTG resists computational attacks b drawing upon a large "pool of entropy" that is unknown to attackers. Its design significantly obscures each cipher grid instance's configuration details even when operation of the OTG system itself is known.

6. **Everything about the OTG system can be fully disclosed.** The design of the C system compliant with Kerckhoff's Principle, which states that: "The security of a cryptosystem must not be dependent upon the nondisclosure of the algorithm; it s only depend upon the nondisclosure of the key." Everything about the design and operation of OTG is disclosed here. Nothing is kept secret and nothing needs to be attackers gain nothing that might help them to crack any user's password sets.

- **Compatible**

  If the Cipher produced a password that a website could not accept we'd be in trouble. If we were simply making up a password, we would adapt what we've made up to the website's rules and restrictions and store the result. But the OTG system's goal is to free us of having to remember anything. We solve this by using the system's Grid to generate a secure lowest-common-denominator password, and additionally provide a means for optionally adding special characters for those websites which allow them. By restricting the system's output (unlike the somewhat extreme examples above) to upper and lowercase alphabetic characters, we obtain maximum compatibility with **all** websites — with

sufficient security provided by the password's length. (At one hundred billion attempts per second, 127 years to crack.) While that doesn't provide as much security as including special characters, this provides compatibility with the great number of sites that still do not allow the use of any special characters. And adding special characters is also explicitly and fully supported by the OTG system. (And note that any other password generator would face the same limitations while providing a less user-friendly solution.)

- **Variable security**
  Today's websites have differing rules for password length. For reasons of their own, some sites place lower and/or upper limits on the lengths of the passwords they will accept. So any robust password ciphering system should be able to generate domain-name-based password strings of any length. The OTG system can.

- **Flexible & powerful**
  A flexible password generating system should be able, when needed, to generate "alternate" passwords for a given domain name. This might be useful if the primary password is compromised, or if a site's password policies requires its passwords to be changed periodically. A system that cannot generate alternate passwords on demand would prove too limited. OTG readily produces up to 52 unique and completely different passwords for every domain name.

- **Everlasting & future proof**
  Technology moves at a rapid pace, often obsoleting older technologies. How many people are still able to play 33 RPM vinyl record albums or 8-track tapes with the equipment they have in their homes or in their cars? Sometimes a solution that does not depend upon technology is superior because you'll **never** find yourself unable to "play" it. Because it is just a carefully constructed piece of paper, the OTG system can be used fifty years from now if we're stuck using passwords until then.

- **Utterly reliable**
  The OTG system requires no instrumentality, uses no power, no batteries, no computers. It cannot be rendered obsolete by some future upgrade. You won't be without it if a web browser upgrade no longer supports its earlier plug-ins (as does often happen), or if the company providing a free service starts charging for it, is acquired by another company, or goes out of business. It's yours forever. There's nothing to break.

- **Universal**
  Since OTG can encipher ANY textual input into a deterministic string of encrypted output, which can be readily recreated at any future time, it can be used for any other purposes as well. For example, if a file must be encrypted with a secret key known only to the sender and one or more receivers, all parties could have an identical instance of a single OTG grid used just for this purpose. The sender encrypts a (plaintext) string of six to eight characters, which is then used as the key to encrypt the file. The user sends the file along with the plaintext string **in the clear** without any concern about eavesdroppers. Only the recipient with the same instance of the OTG grid can take the plaintext string and encipher it into the same key to decrypt the file.

See the next page "How to use the OTG system"
for an understanding of how the system works.

Off The Grid Resource Pages:

GRC's Ultra-High Entropy Pseudo-Random Number Generator

Jump To Top

Last Edit: Aug 16, 2011 at 10:30 (473.47 days ago) Viewed 3 times per day

WORK IN PROGRESS

Off The Grid

How to use the OTG grid

WORK IN PROGRESS

How It Works

Memory:

In order to support several of its important security features, such as its ability to have **all** characters of the input affect **all** characters of the output, the OTG system must have some form of "**memory**". In other words, its future must be affected by its past. In computer jargon we would say that it must be "stateful", or able to "have state."

past state affects current state...

State:

The OTG system has a finite number of states. In fact, it has exactly 676 (26x26) states, and it can be in exactly **one** of them at any time. In computer jargon the OTG system would be described as a **finite state machine.**

Location:

How do we create a simple system having "memory" and "state"? We do it by using a grid on a piece of paper, where at any point in time we are **located** at one specific position on the grid.

But let's back up for a moment since we're getting ahead of ourselves. Consider this (simplified for clarity) grid containing a very special arrangement of lowercase alphabetic characters:

g	e	a	m	o	n	z	k	i	r	c
k	a	n	c	m	z	o	r	g	i	e
n	k	c	z	a	m	r	g	o	e	i
z	o	i	a	n	g	e	c	r	k	m
m	r	z	n	g	a	k	i	e	c	o
a	g	e	i	z	r	n	o	c	m	k
r	c	g	k	e	i	m	n	z	o	a
e	n	k	g	i	o	c	z	m	a	r
i	m	o	e	r	c	g	a	k	n	z
c	i	r	o	k	e	a	m	n	z	g
o	z	m	r	c	k	i	e	a	g	n

Even after studying it for some time it probably looks rather random. It actually **IS** very random (which is a good thing for our purposes). But it was also deliberately and carefully designed by some clever software to have exactly **ONE** very important property. Can you see what it is? If you have a theory, or if you give up, click the button below to add some highlighting to just the 'a' characters and study their relationship.

Toggle 'a' Highlighting

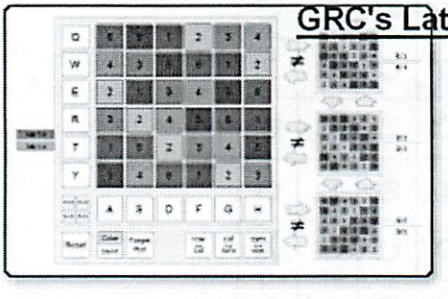
Do you see what's so special?

What's very special about the grid of characters above, is that **EVERY** character, not just the 'a's, appears exactly **ONCE** in every row and column of the grid. This special grid organization is called a Latin Square *(see wikipedia if you're curious to know more)* and it lies at the heart of the operation of the OTG system. In computer jargon we would say that the OTG system is **a finite state machine defined by a Latin Square.**

LATIN SQUARES

Although knowledge of squares having this special property of "*exactly one of each symbol in each row and column*" extends as far back into history as medieval Islam, their systematic analysis was not undertaken until 1779, by the Swiss mathematician and physicist Leonhard Euler (pronounced: "Oiler"). Euler dubbed such grids "Latin Squares" because he used Latin language characters to populate their cells. Amazingly, the many fascinating properties of Latin Squares have occupied mathematicians ever since, and continue to, even today. For example, even today with all of the mathematical analytical capabilities and computers at our disposal, we **only** know how many different Latin Square configurations can be made from grids up to 11x11. **No one knows** how many Latin Square configurations can be made from grids of 12x12 or larger! No one. This is partly because the number is so large. What **is** known, however, is the lower-limit for the number of Latin Squares of various sizes. See the OTG Security Analysis for more information.

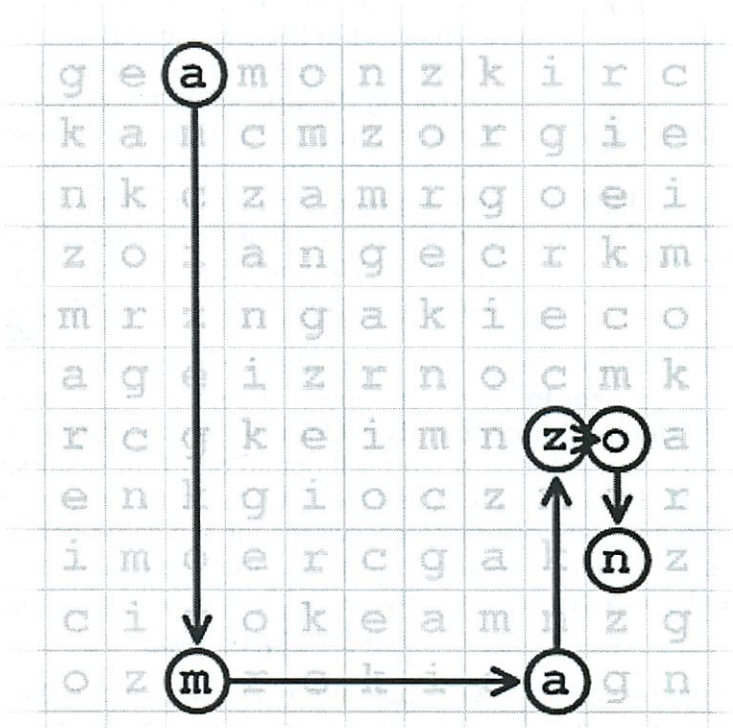
Oh wow — for Miguel to read



GRC's Latin Squares Workbench

During the early development of the OTG cipher system, we needed to develop, prove, and demonstrate a number of theories of Latin Square manipulation (in order to make them for ourselves). To aid in this research we created an interactive "Latin Squares Workbench" to allow for the comfortable manipulation and experimentation with Latin Squares from 3x3 to 6x6. You are invited to experiment with these intriguing mathematical constructions which have fascinated mathematicians for centuries.

Why do Latin Squares matter? . . . Because they allow us to do this:



Note that for illustration purposes we are using a reduced size 11 by 11 Latin Square, containing only 11 of the 26 lowercase characters of the English alphabet. The actual Off The Grid system uses a full size 26 by 26 grid.

Toggle Legend

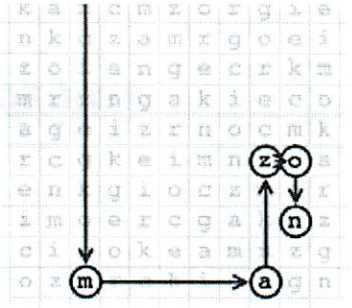
THE KEY PRINCIPLE of the "Off The Grid" cipher is that given a 26x26 Latin Square containing the 26 characters of the Roman (English) alphabet, a domain name can be used to direct a unique path through the Square, where the path taken is determined by the domain name **and** the Latin Square's specific configuration.

This was the key breakthrough I had when I was working to invent a means for creating "memory" and "state" for a domain name to password encryption system: If every row and column has exactly one of each character, then it is possible to move throughout the grid, alternating between movement along rows and columns. In each case, moving to the next character that occurs in the domain name. In this fashion our location depends upon the history of all previous characters, and where we end up is determined by every character.

The use of the Off The Grid cipher system proceeds in the following two separate phases:

Phase 1: Trace a path to the Phase 2 starting point

The OTG system employs two "Phases" for the



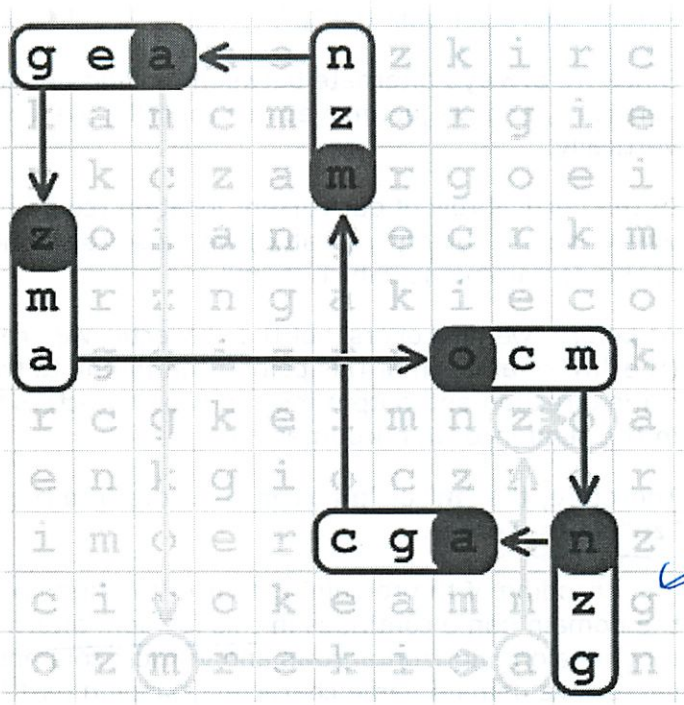
encryption of a domain name into a secure domain-specific password: The first phase determines the starting location for the second phase by tracing the first six of the domain name's characters through the Grid, as shown to the left and (larger) above. Once the first phase has determined the starting point, the second phase is used to emit the enciphered password characters. (We answer the question "Why only the first six characters of the domain name?" on the Tech Details and FAQ page.)

As shown in the diagram, each step alternates between following along a column or a row. Although you **could** start from any of the Grid's 26 columns, or any of the Grid's 26 rows, the most important consideration is consistency. So choose a method and stick to it, otherwise you will obtain different results from one use to the next. But, at the same time, this flexibility can come in very handy. If you **should** need to generate alternate passwords for the same domain (such as when a domain's password policies require that passwords are changed), a total of 26+26, or 52, different passwords are readily available simply by starting in a different row or column.

The general rule for standard OTG operation is to start along the top row, locating the first character of the domain's name there, then finding the domain's second character in the column below . . . and so on. You should study the larger diagram above for the six-character domain name "amazon" until this makes sense to you. (Remember, this is only a reduced-size demonstration OTG grid. The OTG introduction page contains a full randomly generated 26x26 Latin Square for reference if you want to practice tracing out other domain names.)

Phase 2: Trace another path while outputting characters

The second phase of the encryption process very similar to the first, but with a few additions:



just coincidence that it comes back

"amazon" enciphers to "gcznegmacmzg"

As shown in the simplified diagram above, the second phase path **BEGINS** where the first phase path ends. In this way, the path traced during the first phase determines the starting point for the second phase. Therefore, the first six characters of the domain name participate in determining the starting point for the domain name's encoding during Phase 2. The row or column where the phase one path ends is the location where the first character of the domain name is found to begin the second phase's path.

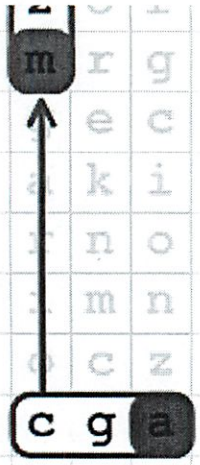
During Phase 2, we use an "overshoot method" to select the two enciphered output characters for each single domain name input character. If you examine the diagram above you can probably figure this out for yourself.

Starting Phase 2 where Phase 1 left off, at the final "**n**" of "**amazon**", we move horizontally (since just previously we had moved vertically to reach the "**n**") to the first character of the domain name "**a**". We then **continue moving in the same direction**, skipping over the character we're seeking, and **outputting the next two characters we encounter, in order, along our direction of travel.**

oh go twice! what does that do

Thus, the first "**a**" of amazon encodes into "**gc**" since, after skipping past the "**a**" we first encounter a "**g**" followed by a "**c**".

Next, having skipped two characters past the first "**a**" character of "**amazon**", we are now **located** at the "**c**" character, the



second character past the "**a**".

Since we **always** alternate between moving horizontally and vertically, and we just moved horizontally, we now move vertically.

In the vertical row of our current location (containing the "**c**" where we are currently located) we move to the next character of the domain name, which is "**m**".

As before, we now overshoot the character we are moving to (the "**m**" of amazon at this point), continuing to travel in the same direction and outputting the two characters that follow it, in the sequence we encounter them.

Therefore, the "**m**" of "**amazon**" encodes into the two characters "**zn**", in that order.

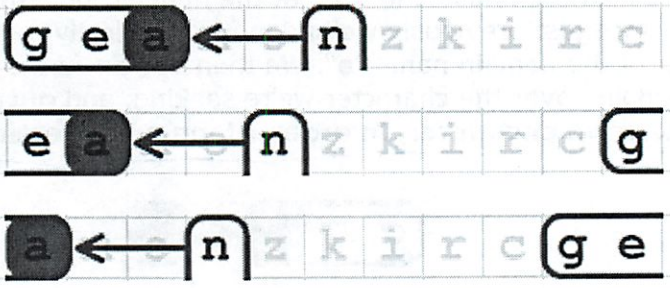
The encoding of the remaining four of the first six characters of the domain name continues in this fashion, alternating between moving along horizontal rows and vertical columns, finding each successive domain name character, skipping past it, and outputting the two characters that follow in the sequence they are encountered.

Before proceeding, please review the Phase 2 encoding diagram above to make sure that the procedure described so far makes sense to you.

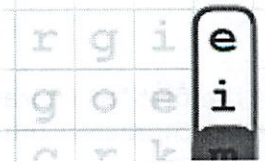
A Few Details:

1. Since Phase 2 enciphering requires that we "overshoot" our target character, outputting the two characters that follow it, it is possible that there might not be any characters to follow if our target character is either on the edge or just one character away from it. When we encounter that situation we "wrap around" to the opposite edge of the grid, in the same row or column, to find the one or two characters past our target.

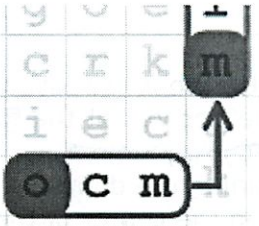
Most of the time, the two characters past our target will be right next to it. But... When the target character is too close to an edge, the two output characters will be found by "wrapping around" to the other side of the grid.



When movemen off one edge, we the opposite sid same row or col

2. 

Imagine that instead of enciphering the domain name "**amazon**" we wish to encipher the domain name "**amazom**" (ending in the letter "**m**"). As we



can see from our example grid, this is a bit tricky because after encoding the "**o**", skipping past it and moving across the two following characters which we output ("**cm**") we are left at the location of second character, which is an "**m**". But now we need to encode an "**m**". Since the grid we're using is a Latin Square, we know that the column we're in only contains one "**m**" character, and we're already standing on it, so we cannot move along the column to it.

The solution is simple: We move one more character in the same direction we were moving to the next row or column (wrapping around if this would cause us to fall off the grid) and locate the character we wish to encode in that row or column. As we can see from the example to the left, the final "**m**" of "**amazom**" would encode into "**ie**".

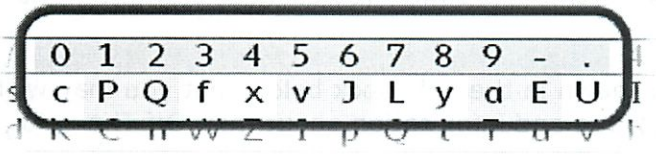
3

Mixed UPPER and lower case: For the sake of simplicity and clarity, the examples we have shown above have used only lowercase characters. The actual OTG system mixes ~~upper with~~ lower case characters to create much stronger (4096 times stronger) passwords than if its passwords were all either upper or lower case.

The rule for using the OTG mixed-case grid (see the sample to the left) is to simply ignore the case of the grid's characters when looking up characters, and output the character's upper or lower case when recording the enciphered output.

Note that the grid's horizontal lines and vertical coloration has no meaning of its own. Those are merely intended to serve as dividers to aid the user in keeping their place when scanning horizontally across the rows or vertically up and down the columns.

4



Numbers, hyphens (-) and dots (.): Internet domain names may contain numbers, hyphens, and separating dots. But if these additional twelve characters were included in the main body of the grid, giving them weight equal to the 26 alphabetic characters, the grid would be expanded from 26 by 26 to 38 by 38, and the grid's area would be increased from 676 cells to 1,444, more than doubling the grid's size. Mixing these twelve seldom appearing characters into every row and column of the 26 often appearing characters would increase the difficulty of finding the characters most often being searched for.

These seldom used characters can be accommodated without increasing the grid's size by creating a simple "substitution table" located at the top center of every OTG grid. (See the sample above/left.) Any time a number, a dash, or a dot is encountered when examining the domain name's first six characters, that character is located in the center of the top border, in the grey region and the **alphabetic** character appearing immediately below it is used instead.

Referring to the sample above, the digit '0' appearing in a domain name would be treated as if it were the letter "**c**" so a "**c**" would be searched for in the current row or column. Similarly, if a name-separating dot (.) character was encountered, it would be treated exactly as if it was a "**u**" while tracing the domain name's path.

That's The System.

What does it all mean?

You should now understand how the "Off The Grid" system operates. Using an OTG grid, the instructions above enable you to securely convert any string of characters, such as from an Internet domain name, into twice as many mixed-case alphabetic characters for use as that domain name's corresponding password. You need not write down the password if you choose, since you can easily recreate the same password from the same domain name at any time in the future.

If the recommended six input characters are used, a new password string consisting of twelve mixed-case characters will be produced from out of **191,102,976,000,000,000** different possible 12-character passwords. Since "bad guys" have no way of determining which one of those **your** own personal grid produced, the result is **very** good security. Even so, you may wish to consider some enhancements to further increase the system's security: Check out the Enhanced Security Options page. But either way . . .

Once you have used the **Create/Print Your Own Grid** page to create your own personal custom grid (**which NO ONE ELSE will ever have**) you will have everything you need to create secure domain-based website passwords using a simple "no tech" paper-based system that needs no batteries, can't be hacked, can be used forever, will never be incompatible with web browser upgrades, and on and on.

There are a number of pages shown in the link block below that you may wish to examine for additional background and information about this "Off The Grid" system:

Off The Grid Resource Pages:

1	"Off The Grid" Introduction	**6**	Frequently Asked Questions
2	Security Goals and Design	**7**	Technical Details and Docs
3	How to use the OTG system	**8**	Security & Attack Analysis
4	Enhanced Security Options	**9**	Latin Squares Workbench
5	Create/Print Your Own Grid	**10**	"Off The Grid" Feedback

WORK IN PROGRESS

Off The Grid

Options for Enhanced Security

WORK IN PROGRESS

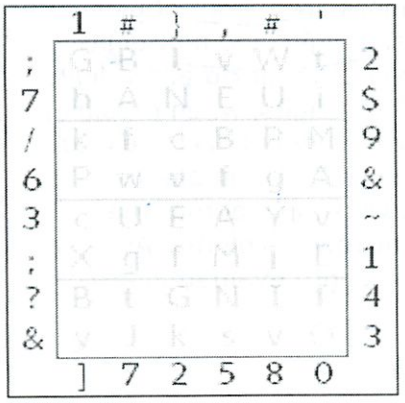
In security system design, it's impossible to be too paranoid.

No security system deserves to be trusted without a careful and critical examination of that system's potential vulnerabilities. For that reason, the Off The Grid security and attack analysis page examines the system's strength against various forms of active attack.

During the development and extensive discussions of the system, a number of possible enhancements were tossed around, examined, explored and considered. Although they **do** further strengthen the system in various ways, we concluded that on balance the simple base system, as described on the How to use the OTG system page, was already so strong that the additional strengthening didn't return enough to justify the additional difficulty.

However, since we did develop the extra strength approaches, it's worth sharing what we came up with in case they might be something **you** would like to consider. As we wrote above . . . it's impossible to be too paranoid.

Adding Non-Alphabetic Characters



To the security aware password user, especially any user who is already familiar with GRC's other password-related work, including our Password Haystacks password strength evaluation system, the single biggest surprise might be that the standard Off The Grid system does **not** incorporate the generation of passwords containing non-alphabetic digits and other special characters.

During the development of OTG, we experimented with designs that **did** generate passwords having much larger character sets **[see: 26x26 & 13x13]**. But the trouble, aside from making the final grid much more noisy visually (and no way around that), was that there was then no way **not** to generate passwords **without** special characters. You got what you got. And that would be a problem any time a website refused to accept passwords containing special characters. A recent analysis of websites showed that a surprising percentage still refuse to allow complex passwords. So we needed another approach.

You may have noticed that the grid shown on the OTG introductory page contained an inner Latin Square region surrounded by a border of digits and special characters. As with the grid's inner composition, an important non-obvious property governs the design of that border: every row and column has a digit at one end and a special character at the other, and all of those border characters are randomly chosen and arranged, with the exception of the ten digits, the dash and dot at the

center of the top border (which form the translation table). And even there, the rule of a digit and a special character at each end holds. Thus, as you are moving through the rows and columns of the grid, randomly chosen digits and special characters are continually available to you for whatever purpose you might wish.

There are no hard and fast rules for their use, but here are some example strategies for incorporating them into the resulting password:

- Letting the grid **CHOOSE:**
  The password encoding scheme defines a current row or column and a direction of travel. This means that there is always a "border character" out at the end of the current row or column in the last direction of travel. So a potent scheme for embellishing OTG passwords with grid-chosen digits and non-digit special characters is to output the two characters following the target character as usual, then scan in the same direction to the end of that row or column out to the border and output whatever character is found there. This would generate passwords of the form **<alpha><alpha><non-alpha>** with three characters output for every domain name character input. Since eighteen characters might be longer than some sites allow, you might opt to reduce the OTG path lengths from six down to five or perhaps even four. This would result in passwords of fifteen or twelve characters respectively, but incorporating a great deal of entropy due to their inclusion of randomly chosen digits and non-digit special characters.

  (Remember that you can quickly and easily use our Password Haystacks page to quickly evaluate the attack resistance of any password format you are developing.)

- Letting the grid **HELP:**
  As detailed above, letting the grid choose is terrific as long as the target website allows the use of non-digit special characters. The chances are 63 out of 64 (98.4%) that a password containing six randomly chosen digits and non-digits will contain at least one non-digit. Unfortunately, some sites that **do** allow digits **don't** also allow non-digit special characters. Since adding digits, when possible, is better than not adding anything at all, we can easily modify our "Let the grid choose" approach to "Let the grid help": Since each row and column always contains one digit and one non-digit special character, we simply output the numerical digit out at the one of the ends of the current row or column. As before, the same comments about overall length and strength evaluation are relevant here.

- Tack something onto the **END:**
  A third possibility that represents a useful compromise would be to simply tack on the final row's digit and non-digit special character, or perhaps just the final numerical digit. As the Password Haystacks page demonstrates, this significantly strengthens any alpha-only password while not increasing its length significantly. You could use the border character pointed to by final row traversal, or just always the digit that appears on that ending row.

Are special characters necessary?

Do we need more security?

To answer that (↑) question, let's look at the complexity of the character sequences generated by the standard OTG grid system: As we know, each domain name input

character generates two password output characters. Using the "overshoot" method, neither of the output characters can be the same as the domain name character. So that means that the first output character could be any one of the remaining 25 alphabetic characters and the second output character could be any one of the remaining 24 characters. Since each could be either upper or lower case, this yields any one of 50 possibilities for the first and any of 48 possibilities for the second giving us a total of 2400 possible character pairs output for every character input.

Assuming that we use six characters from the domain name, that yields 2400 raised to the power of six (2400^6) possible combinations, which is exactly 191,102,976,000,000,000,000 possible twelve-character (six character-pair) passwords. (You can use our web based "Big Number Calculator" to confirm this for yourself if you wish.) If we assume that an online attacker is able to try 1,000 guesses per second, with 31,557,600 seconds per year, then 6,055,687,885 years would be required to try all possibilities. A million guesses per second reduces that to 6,055,688 years. And a billion guesses per second drops it all the way down to approximately 6,056 years.

Therefore, while it certainly **is** possible for us to increase the strength of the standard alphabet-only OTG system by adding non-alpha characters, the twelve random alphabetic characters produced by the standard use of the grid provides ample strength. And, if you stick with alpha-only, you don't ever need to worry about whether websites will accept more complex passwords.

Note that if a website requires the use of at least one digit, as some do, you could always simply add one somewhere. Even if you were to always do the same thing, even if your approach was known to an attacker, you would still have at least as much strength as you do without using any digits . . . which we have already seen is more than sufficient.

If you wanted more security WITH full compatibility...

You could also have that easily by simply encoding more of a domain name's characters. Even the very short domain name "GRC.COM" consists of seven characters which could be encoded, and most domain names are longer. So there are usually plenty of domain name characters to encode. As we have seen, each additional character-pair added increases the attack strength by 2400 times. So simply using seven domain name input characters to produce 14 alphabetic-only output characters yields 2400^7 = 458,647,142,400,000,000,000,000 possible combinations, which would require **14,533,651 years** to test at one billion attempts per second.

Thus, the simplest and most universally compatible means of increasing the system's security is to make your passwords longer. (Though we think that 12 output characters is already very good security!) But if you do want to embellish your passwords with non-alphabetic characters, the Off The Grid system makes that possible too.

Further obscuring an individual grid's structural details

The Off The Grid security and attack analysis page examines the OTG system from the perspective of an attacker who might somehow obtain one or more (or many) secret domain name and OTG password pairs of a given user. On that page we attempt to determine to what degree the system's security could be compromised in such a way that additional passwords might be determined for other domain names.

In other words, how much of a grid's secret structure is revealed by an attacker knowing the specific passwords generated by specific domain names? Please see that page for the details of that analysis. However, as you probably expect, this entire OTG system would never have been disclosed and published if the system hadn't survived that analysis with plenty of margin to spare. (It did!) In fact, so little useful information about the structure of any individual user's OTG grid is leaked when existing domain names and their associated passwords are discovered that everyone who participated in that analysis is comfortable with using the system as it is, without additional enhancement(s).

But you may also know by now that I always want to explore all possible avenues for improvement . . . because you never know what you might find until you look.

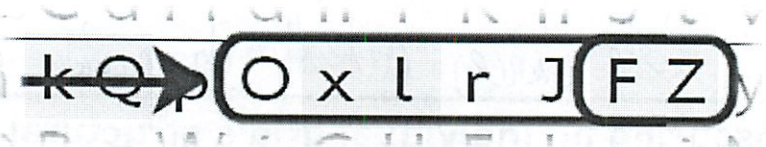
So here's the theoretical structural leakage concern: An attacker who obtains one or more matching domain name and password pairs generated by an OTG grid obtains multiple three-character "triples" of character sequences occurring in the user's grid. The attacker knows the first character of each triple, since it comes directly from the domain name. And the attacker knows the two characters that lie directly adjacent to it on the grid, since those are the two characters of the output. By assembling each character of the domain name with each character pair of the associated password, the attacker can generate a set of "triples" which are known to occur somewhere within the user's grid. As you can see, this reveals pieces of the composition of a user's OTG grid. As mentioned above, the security and attack analysis page examines what an attacker might be able to do with one or more sets of these triples, i.e. is this a problem?

However, here we will examine a means of hiding even that (small and unusable) amount of our grid's composition.

We've already seen that every OTG grid has a Latin Square in its center, surrounded by a border of carefully designed special characters such that every row and column always contains one of each. Every OTG grid has one additional non-obvious property: In any direction of travel, up, down, left or right, including wrapping around an edge, continuous runs of the same capitalization are limited to three characters. If you look back at the grid on the introduction page you'll see that nowhere, in any direction, are there ever more than three characters together all with the same case. The capitalization is still locally random and unpredictable, but it also has this significant property.

Okay . . . so??

This special property makes it practical to enhance the "overshoot" method of finding our output character pair: After encountering the target character we are scanning to, instead of taking the next two characters adjacent to it while travelling in the same direction, we hold off until we have first encountered one of each alphabetic case of intervening characters.



In the example above, we were scanning to the right for the next domain name character "**o**", which we found. Then, instead of outputting the two characters "**xl**" that follow, we continue scanning until we have encountered at least one lowercase character and at least one uppercase. In this example, the first "**x**" we encounter satisfies our search for a lowercase character, and the uppercase "**J**" satisfies our

search for the first uppercase character. With both searches satisfied, we then output the **next pair** of characters encountered, which is "**FZ**".

The power of this approach is that it "repurposes" the grid's variable capitalization in a way that cannot be known to an attacker. Since we wait for at least one of each case before outputting, and since continuous runs of the same case are limited to three characters, the two characters that are eventually outputted are always spaced between two and six characters away from the domain name target character. And since we don't output the characters we are skipping over, their case information is not contained in our output and attackers can never know what the spacing was. The three letter "triples" are thus broken up so that the attacker no longer gets this information.

How is this theoretically better

It's true that this scheme still reveals pairs of characters that are adjacent since successive output characters are adjacent. If we wished to eliminate even that, we could output only one character after encountering both cases, then output the second character only after again encountering both cases. This would completely break up the "triples" with an unknowable number (2 to 6) of separating characters.

Harder to reconstruct

But once again, as the security and attack analysis page demonstrates, **none** of these additional enhancement measures are necessary for the Off The Grid system to deliver extremely good security. They were presented here for completeness, because they were developed and considered during the development of the OTG system and ultimately proved to be unnecessary. The simpler the system is to use, for more it will be used. The creation of a usable system was our goal.

Off The Grid Resource Pages:

1	"Off The Grid" Introduction	**6**	Frequently Asked Questions
2	Security Goals and Design	**7**	Technical Details and Docs
3	How to use the OTG system	**8**	Security & Attack Analysis
4	Enhanced Security Options	**9**	Latin Squares Workbench
5	Create/Print Your Own Grid	**10**	"Off The Grid" Feedback

GRC's Ultra-High Entropy Pseudo-Random Number Generator

Jump To Top

Last Edit: Aug 21, 2011 at 08:40 (468.55 days ago)

WORK IN PROGRESS

Off The Grid
Frequently Asked Questions

WORK IN PROGRESS

Q: *Is OTG as secure as the AES/Rijndael symmetric encryption system?*

A: The two systems cannot be readily compared because they are very different. However, that's really not the right question to ask. (See the next Q&A question if you insist, but read this answer first.)

It was never the goal of our "Off The Grid" project to duplicate the security of a state-of-the-art symmetric cipher like AES/Rijndael. Rather, the OTG system was carefully and deliberately designed to provide **far more security** than any of the solutions it replaces, while being easy enough to use that it would actually be used. We believe the system succeeds at achieving **those** goals.

It is important to consider that modern computer-based cryptography is so strong that it is never the weakest link in the chain. It is never broken. Yet Internet user's passwords are still being lost or stolen, and their accounts are being compromised every day. The problem is not that computer-based encryption is not strong enough — it's effectively unbreakable. The problem is pretty much with everything else. Users choose and use insecure and easily guessed passwords, and they often use the same password on every, or across many, sites. Malware and keystroke loggers of many types can infect users' systems, and popular password management systems have been the target of malicious manipulation in the past. But users who adopt the "Off The Grid" system have a paper-based solution that does not run in their computers, so it cannot be compromised by malware. And since all OTG passwords are long, extremely random and unique per site, OTG provides the best security we know how to create. OTG is far more secure than the other "ad hoc" methods that most users adopt. So switching to OTG almost certainly increases any user's security.

Q: *But I want you to compare the security of OTG to AES/Rijndael . . .*

A: Okay.

First: Algorithmic Security — Modern cryptographic algorithms utilize a high number of complex computational and combinatorial steps to obscure the relationship between the input "plaintext" and the output "ciphertext". And even with a modern processor's incredible speeds, pushing data through these algorithms takes some time. There does not appear to be a feasible way to adapt those "number crunching" approaches to a simple-to-use paper-based cipher. So we needed to take a different approach . . .

Second: The "Off The Grid" system replaces symmetric cryptography's computational complexity with extremely high levels of entropy (unknowable and unpredictable randomness). The AES/Rijndael cipher is most commonly used with a 128-bit key. This key length provides 2^{128} possible keys which is: **340 282 366 920 938 463 463 374 607 431 768 211 456**. The

extreme algorithmic strength of the AES/Rijndael cipher prevents it from "leaking" any useful information about its key when both its input and output are known, and, as you can see, there are too many possible keys for it to be practical to try them all.

Since a manual system like OTG cannot have such computational complexity, we compensate for that by incorporating a truly incredible amount of entropy (randomness). The entropy of Latin Squares is so large that no one even knows how large it is! Mathematicians have established a "lower bound" for it. This means that while they don't know the exact amount, they know that it has at least a certain amount. How much? The OTG Security & Attack Analysis page carefully explains where the following number comes from, but there are known to be at least **9 336 974 347 720 076 203 095 381 302 683 075 484 706 012 030 875 383 265 106 777 232 515 384 291 786 329 470 875 840 456 766 821 029 030 235 438 914 174 291 844 167 774 650 650 291 329 460 401 751 489 013 555 810 781 700 163 431 985 765 122 298 613 958 200 230 192 236 631 943 316 085 768 502 914 719 815 963 609 471 283 139 690 899 669 496 766 419 404 467 151 772 248 428 431 825 394 305 641 480 706 711 487 437 686 906 450 684 680 968 293 622 304 401 609 062 321 217 193 606 241 756 724 745 170 796 786 016 394 203 303 300 168 583 550 145 590 123 023 289 449 057 087** possible 26x26 Latin Squares of the size used by, and randomly chosen by, our Off The Grid system. Expressed in scientific notation, this number is: 9.337×10^{426}. The log(2) (logarithm base 2) is approximately 1418. So it is the equivalent of a 1418-bit key. And the number is even higher because the random alphabetic case of those 676 characters further increases the grid's total entropy.

Unlike a state-of-the-art symmetric cipher, which does not "leak" any useful information about its key when it is used, **every use of the OTG grid does leak a little bit of information about its key**. This occurs because the OTG's key **is** the specific structure of its grid, and we output selected pieces of the grid as the OTG passwords. This means that individual websites each obtain tiny chunks of the OTG grid structure selected by their domain name. But look again at the phenomenal number of possible OTG grids, any one of which any single user might be using. Our careful analysis demonstrates that so little useful information can be obtained from each OTG domain name and password pair that even if an attacker were to somehow collect a large number, there are so many possible OTG grids that nothing useful could be done with that information.

So, as you can see, modern symmetric ciphers, with their emphasis on computational complexity, employ sufficient entropy (typically 128 bits) to thwart attackers by not leaking anything about their key. By contrast, because computational complexity is not feasible in a simple manual system, OTG takes advantage of its incredible amount of entropy (a user has **one** of more than 9.337×10^{426} possible Latin Squares) to mitigate the consequences of a bit of its "key" leaking to anyone who obtains an OTG password.

Can you say anything more specific?

Q: *Why are there different ways to use the system?*

A: Unlike many of the "black boxes" in cryptography, where something is put into one end and something different mysteriously emerges from the other

end, the OTG system is, in every way, open and transparent. It should be easy for anyone to understand and use. These pages carefully present the motivation, goals, and design of the system. And like any open system, users are free to use it for any purpose and in any way they choose. Once the OTG Latin Square with its special border characters and capitalization rules is understood, its special properties can be applied in many ways. Users should be, and are, free to experiment, explore and find the approach that suits them best. It was never our intention to impose any strict usage upon anyone.

Q: *Why do you have one domain name character expand into two password characters?*

A: This appears to be the best tradeoff. You could change it to one in and one out, or one in and three out, or ven alternate . . . or anything else you want. But we know that good security requires in the neighborhood of twelve variable-case alphabetic characters. Since the standard OTG system encrypts six characters into twelve, even the shortest domain names, such as ab.com or abc.tv, provide sufficient raw material for OTG's encryption.

Q: *Could I use this system for other things too?*

A: Absolutely! For example, if two people each shared a common OTG grid for the purpose of exchanging encrypted files, a file sender could choose a simple word, and use the OTG system to encrypt that word into twice as many password characters. Then that password could be used to encrypt a file to be sent through eMail. The eMail message could contain the pre-encrypted short word that was used, along with the encrypted file. In fact, the key word and the file could even be posted publicly.

The recipient would receive the eMail, run the short word through their identical copy of the OTG grid to obtain the same encryption password that was used to encrypt the file, and decrypt the file. And, of course, many other uses are possible.

how share grid - - -

Q: *I don't think my mother-in-law could understand or use this.*

A: We certainly understand that this system is not for everyone. It will appeal to people who like the idea of generating secure passwords with a system that cannot possibly be compromised by malware or other technologies . . . because it doesn't rely upon computers for its operation. But in working without computers the OTG system inherently and necessarily places responsibility for its operation on the user. There's no way around that. The system was designed to be as simple to use as possible while still offering an extremely high level of security. It achieved its goals, even though those goals don't and won't make sense for everyone.

lol

Off The Grid Resource Pages:

1 "Off The Grid" Introduction **6** Frequently Asked Questions

GRC's Ultra-High Entropy Pseudo-Random Number Generator

Jump To Top

Last Edit: Aug 22, 2011 at 10:40 (467.47 days ago)

Viewed 2 times per day

WORK IN PROGRESS

Off The Grid

WORK IN PROGRESS

Technical Details and Documentation

Hmmmm

I don't recall now, what I was planning to put here. It may be that things I planned to say have already been said elsewhere, since once I finish the Security & Attack Analysis page I believe that everything will have been described.

I'll leave this page linked and present as a placeholder and reminder in case something occurs to me as I'm wrapping everything up.

Off The Grid Resource Pages:

1	"Off The Grid" Introduction	**6**	Frequently Asked Questions
2	Security Goals and Design	**7**	Technical Details and Docs
3	How to use the OTG system	**8**	Security & Attack Analysis
4	Enhanced Security Options	**9**	Latin Squares Workbench
5	Create/Print Your Own Grid	**10**	"Off The Grid" Feedback

GRC's Ultra-High Entropy Pseudo-Random Number Generator

Jump To Top

Last Edit: Aug 22, 2011 at 10:59 (467.45 days ago)

Viewed 1 times per day

WORK IN PROGRESS

Off The Grid

Security and Attack Analysis

WORK IN PROGRESS

This page is currently incomplete.

It will soon contain an analysis of the nature of the information that the OTG system "leaks" due to the fact that its passwords are formed from bits and pieces of the user's grid. This is already discussed at some length in the second answer on the OTG Frequently Asked Questions page. Please see that if you are interested in the security of the OTG system.)

Essentially, every OTG password inherently contains some structural information about the grid. But this page will show that so much remains uncertain and unknowable that even if an attacker were to obtain a great many samples of a user's domain names and matching OTG passwords, they would still be unable to reassemble the user's grid.

(Notes to myself) Discussion Points:

- The need to protect physical access to the grid
- OTG uses high levels of entropy instead of algorithmic complexity
- The attack model is: Bad guy knows that user is using the PPC system to generate passwords and somehow gets one or more of that user's domain & password pairs. Is there any way the bad guy can gain enough useful information to weaken the secrecy of the user's other domain-based passwords.

Reader Beware: Much of what's below refers to an earlier design of the OTG system **[see: 26x26 & 13x13]**, so please do not rely upon any of this information until the page is complete.

ok - we need to redo

How many steps should you take?
If you were using the PPC system to encipher a very long domain name such as "allthingsconsidered.com" the task of following that name's entire path to the end would quickly become quite tedious and prone to error. It is also unnecessary for the security of the Personal Paper Cipher.

As we'll see next, the system's Phase 2 (enciphering) begins at the location determined by the initial Phase 1 path following.

The first question that arises is: How secure is this?

As we will see below, only a **portion** of the Personal Paper Cipher's security comes from the attacker's ignorance of the configuration of the Latin Square we are using. But it does raise the question: could an attacker guess or computationally "brute force" the configuration of the particular Square we are using?

$$(n!)^{2n}$$

The Cipher's design actively prevents the disclosure of any particular Square, so the feasibility of guessing or brute forcing a Square's configuration, assuming the absence of any other clues, is primarily determined by the total

$$n^{(n^2)}$$

number of possible Latin Squares from among which ours was chosen. As was mentioned in the "Latin Squares" box above, mathematicians who have studied Latin Squares have only been able to obtain an exact count for Squares having sizes up to 11x11 (and the exact count for 11x11 Squares was only recently obtained.) What the math gurus **have** been able to state with certainty, is that the **lower bound** on the number of Latin Squares, of a given size 'n' by 'n' is given by the equation shown on the left. In other words, although no one knows exactly how many specific Latin Square configurations there are of any size larger than 11x11 (because there are too many!), we **do** know that there are **at least as many** as specified by that equation. But how many is that?

$$\frac{(26!)^{52}}{26^{(26^2)}}$$

Plugging the number (n=26) into the lower bound equation for the Personal Paper Cipher's 26x26 Latin Square results in a number so large as to be rather ridiculous. It is: *9 336 974 347 720 076 203 095 381 302 683 075 484 706 012 030 875 383 265 106 777 232 515 384 291 786 329 470 875 840 456 766 821 029 030 235 438 914 174 291 844 167 774 650 650 291 329 460 401 751 489 013 555 810 781 700 163 431 985 765 122 298 613 958 200 230 192 236 631 943 316 085 768 502 914 719 815 963 609 471 283 139 690 899 669 496 766 419 404 467 151 772 248 428 431 825 394 305 641 480 706 711 487 437 686 906 450 684 680 968 293 622 304 401 609 062 321 217 193 606 241 756 724 745 170 796 786 016 394 203 303 300 168 583 550 145 590 123 023 289 449 057 087*

This number, expressed in scientific notation, is: 9.337×10^{426} and the log(2) (logarithm base 2) is approximately 1418. This means that a binary number having at least 1418 bits (because this is the known lower bound) would be required to specify one of the 26x26 Latin Squares possible.

During the development of this Personal Paper Cipher system, we developed several different algorithms that randomly found one single Latin Square from among **all** of those that are possible. However, to assure TNO (trust no one) style privacy, this code must run as JavaScript on the user's own web browser (that way no one, not even GRC, has any idea what your individual Cipher Square looks like). The solutions we developed **did** allow for highly efficient client-side JavaScript implementation, but sometimes the search took so long that browser warning messages popped up, and some users have older and much slower systems. In the end we chose a secure compromise that allows us to quickly choose one Latin Square from among a smaller, but still massive subset of all possible Squares.

The final algorithm for Latin Square generation used by the PPC system randomly selects one Latin Square from among a subset containing **$26!^2 \times 25! \times 6$** Squares. This is *15 136 831 721 341 031 145 555 109 377 300 868 031 260 208 035 408 929 173 274 624 000 000 000 000 000 000*; in scientific notation: 1.513×10^{79}, or 263 bits of effective entropy — and security. And, as we'll see below, this is only one of several places from which the PPC system obtains its security.

Off The Grid Resource Pages:

Latin Squares Workbench

Experiments in Latin Square manipulation for the "Off The Grid" project.

Latin Square: An **n** x **n** square array filled with **n** different letters, numbers or symbols, each occurring exactly once in each row and exactly once in each column.

What is this all about?

During the early development of the "Off The Grid" personal cipher system we needed to acquire a working understanding of the characteristics of Square Latin manipulation. It was known from the extensive literature about the characteristics and nature of Latin Squares that if was possible to "permute" any Latin Square in a number of ways which would not alter the Square's fundamental "Latinness". For example, if you think about it for a second you'll see that the "only one of each type of symbol in any row and column property" of every Latin Square will be preserved if any row or column is exchanged with another. In fact, all of the following manipulations preserve any Square's "Latinness":

- Exchange any pair of rows or columns.
- Exchange any set of symbols with another.
- Exchange the columns for the rows (mirror across the major diagonal).
- Exchange the symbol numbers with the row or column numbers (tricky, but it works).

Note that what's **not** here are other, presumed possible, LS configurations that might be unreachable through these transformations (this remains to be determined and is one of the goals of this work). Such "unreachable" Squares might be found, for example, by employing a brute force recursive backtrack searching to find all possible Squares.

(Note that a full recursive backtracking search is what I ultimately determined would be necessary — and is what I'm doing for the "Off The Grid" system — since the JavaScript device below allowed us to demonstrate and explore that it was not, in fact, possible to reach all possible Latin Squares through permutations of a base starting Square.)

This page was created to aid the empirical and experimental exploration of Latin Square generation. Its goal is to answer the question of how complex a "unique scrambling" we are able achieve, and which transformations are useful and which, if any, are not.

It's clear that there are **n!** ways of reordering the columns and **n!** ways of reordering the rows. However, quick experimentation with smaller Squares (such as a 3x3 which can be selected through the user-interface) quickly reveals that the combination of unique color patterns available though combining row and column reordering is not **(n!•n!)** but instead is **(n!•(n-1)!)**.

There are also **n!** ways of swapping symbols (colors) among themselves.

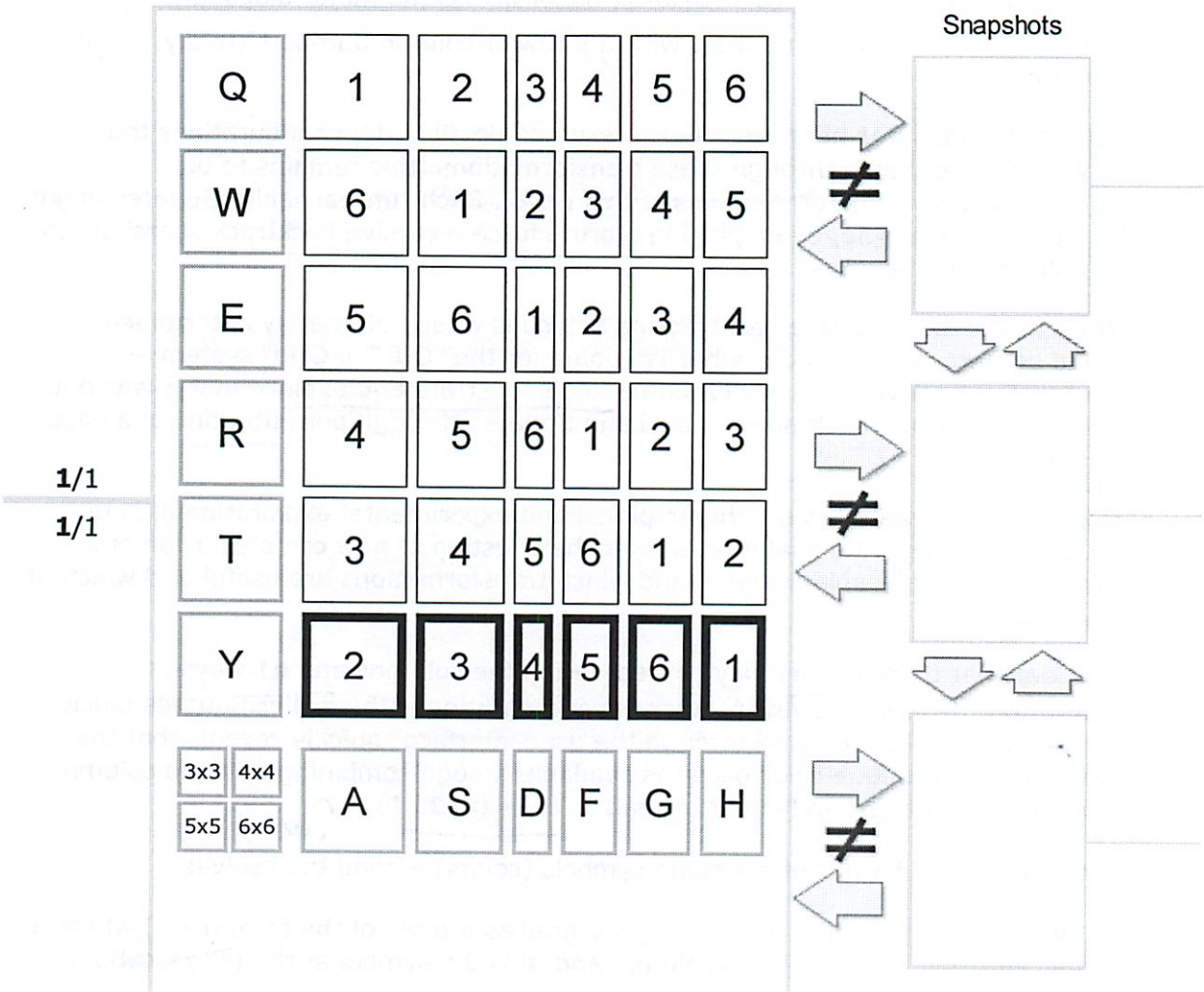
Finally, if each cell in the array is represented as a tuple of the form (r,c,s) where 'r' is the cell's row, 'c' is the cell's column, and 's' is the symbol at the (r,c) location,

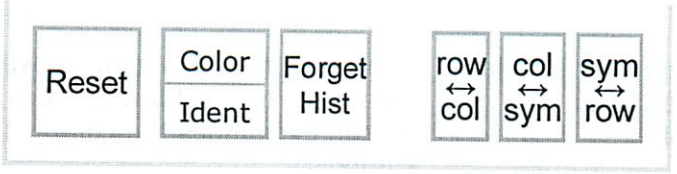
then the entire Square's essential "Latinness" is preserved if any two elements of every tuple are exchanged. For example, if the 'c' and 's' elements were exchanged, the cell's symbol would specify the column and the cell's column would specify the symbol. In other words, the cell's column number would be stored into the column specified by that cell's symbol; e.g. if the cell in column 2 contained a 5, then a 2 would be stored into column 5. The workbench below supports and allows for experimentation with all of these transformations.

Collectively, this generates a potentially huge variety of different Latin Squares. Using just simple row, column and symbol exchanges, we can produce **(n!•(n-1)!•n!)** combinations. For a small order 6 (**n**=6) Latin Square, such as the experimental one below, that's 62,208,000 reordering arrangements.

But here's the BIG question: Are ALL of the Latin Squares resulting from combinatorial rearrangements distinct and unique? And how can we be sure we didn't miss any? For example, in preliminary experiments with the tuple exchanger transformations, alternating between the c↔s and r↔s exchanges cycled through many six intermediate Squares before returning to the starting configuration ~ but only in some starting grid configurations. Might it be that mixing in the otherwise simplistic c↔r exchange would extend this further?

So the question to be answered is: What algorithm can be designed using these transformations to yield the largest number of possible unique Latin Squares? And how large is that number relative to the total number of possible Latin Squares?





Non-Obvious Latin Square Workbench Concepts

The LS Workbench displays the colors of the grid squares, but it also tracks the individual identities of the squares as they move through the grid. In other words, in its default "Color" display mode, all of the Red grid squares show a '1' with the numbers being numeric representations of each color. But in the "Indent" display mode, you'll notice that each color is individually numbered within its color. This is useful for tracking instances where transformations do not result in color changes, but are nevertheless changing the identities of the grid's objects . . . or where extensive transformations may return to a previously seen coloration, but with different instances of the same colors in each location.

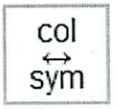
The Workbench keeps track of every grid color and identity configuration it has seen, and it memorizes these individually. The "fraction like" numbers sticking out of the sides of the grid displays show the location of the current pattern in the history memory above the '/' and the total size of the history memory below the '/'. The upper pair of numbers reflects the status of the color-only memory and the lower pair shows the status of the color+identity memory. Finally, the number pairs are highlighted with red or green (respectively) whenever a new and unique pattern entry is made to the history memory

The equality of the snapshot scratchpads is continually shown with equals, not equals, and a green three-bar "equivalence" symbols. Not equals is shown whenever the displayed color patterns are not identical, equals is shown when the displayed colors **are** identical but the underlying identities of the colors are not. And the three-bar "equivalance" symbol is shown when both the colors and identities are identical. Consequently, for example, the three-bar equivalence symbol will be shown immediately after copying the main grid into one of the snapshots since they will be completely identical at that point.

The Workbench is 100% "touch friendly" and is delightful to work with on any touch-enabled device. But it is less easily manipulated with a mouse. Consequently, the Rows, Columns, and Colors can be selected with a keyboard by pressing the letters associated with the rows, columns, and colors.

Workbench User-Interface Guide

The "Reset" button restores the primary Latin Square to its "base" state where the first row is numbered 1, 2, 3, ... and each successive row is rotated one column to the right. This is



If each cell of the grid is addressed as an (r,c,s) tuple, the "Latinness" of the Square is preserved when any pair of those tuple elements are exchanged. Exchanging the 'r' & 'c' elements

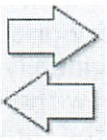
guaranteed to produce a very simple Latin Square and is the starting state for all subsequent manipulations.

swaps the row and column (which is what the row↔col button does). This button exchanges the columns and symbols.

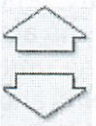
Q The large alphabetic buttons can be clicked to select its respective row or column. The respective keyboard keys perform the same function. The contents of any pair of rows or columns can be exchanged by first selecting one, which will show it selected, then clicking the second, which will exchange the contents of the row or column.

sym ↔ row Similar to the button above, this performs the somewhat complex "Latinness" preserving operation of exchanging the symbol and row elements of the Latin Square's (r,c,s) tuples. Note that 'r' stands for row, 'c' for column, and 's' for the symbol located at the 'r' and 'c' intersection.

1 The Latin Square grid is labelled with distinctive colors and numbers. As with row and column selection/swapping the keyboard may be used for quicker interaction. Any two sets of grid symbols may be exchanged by first selecting one, then the other.

 The Latin Squares Workbench contains three "Snapshot" scratchpads. The horizontal left and right pointing arrows copy the current Latin Square grid configuration back and forth between its respective snapshot and the Latin Square.

3x3 4x4 5x5 6x6 Latin Square complexity increases quickly as the size of the square increases, the workbench can be set to manipulate 3x3, 4x4, 5x5, and 6x6 size Latin Squares.

 The up and down pointing arrows copy the snapshot scratchpads between themselves. This can be useful handy during complex hunts for intermediate Latin Square configurations.

Color Ident This button toggles the workbench's display mode. When set to "Color", the cell numbering corresponds to its color. When set to "Ident" the numbers track the cell's individual identity within its color group as it is moved around the Square.

 The Equals, Not Equals and Equivalence (three green bars) symbols continuously reflect whether each of the snapshot scratchpads is identical to the main Latin Square. "Equivalence" means that not only are all of the colors identical, but the individual cells within a color are also identical.

Forget Hist Empties the Workbench's memory of all previous

30/38 **44**/44 The upper bold number of the n/m fraction is

Hist

color and cell identity patterns. The current pattern becomes the first new pattern in the Workbench's pattern configuration history.

row ↔ col

Flips (reflects) a Latin Square about its major diagonal axis (from upper left to lower right) preserves its "Latinness". (Preliminary experimentation reveals that this reflecting does not help to create unique Latin Squares since the mirroring can be easily reversed through row and column exchanges.

44/44

the associated grid pattern's location within in the Workbench's pattern memory and the lower non-bold number is the total number of pattern entries in the grid memory. The upper pair of numbers shows the color-pattern memory and the lower pair shows the more specific color+identity memory. The respective numbers are highlighted (in red for the upper pair and green for the lower pair) whenever a **new** grid pattern is being seen for the first time and a new entry is made into the pattern memory.

Off The Grid Resource Pages:

1 "Off The Grid" Introduction
2 Security Goals and Design
3 How to use the OTG system
4 Enhanced Security Options
5 Create/Print Your Own Grid
6 Frequently Asked Questions
7 Technical Details and Docs
8 Security & Attack Analysis
9 Latin Squares Workbench
10 "Off The Grid" Feedback

GRC's Ultra-High Entropy Pseudo-Random Number Generator

Jump To Top

Last Edit: Aug 12, 2011 at 13:18 (477.36 days ago)

Viewed 10 times per day

Draft 1 12/2 3PM

Novel Password Systems Where Enter Derivation of Password instead of Actual Password

6.858 Final Project

Michael Plasmeier <theplaz>
Jonathan Wang <jwang7>
Miguel Flores <mflores>

Motivation

The problem with many password systems is that users must type their entire, full password each time they log on. This makes the password vulnerable to key logging and interception during transmission.

We explore systems in which the user does not enter their direct password, but a derivation of the password **which changes on each log in**. The user proves that he or she knows the password without subsequently ever providing the password itself.

ING Password Keyboard

A simple example is ING Direct's PIN pad. Under ING's system, the user enters the letters corresponding to their PIN instead of the PIN itself. The mapping between numbers and letters is randomly generated on every log in. This method does not survive an attack where the attacker has access to the mapping, but it does prevent simple keylogging.

Figure 1 ING's Pin Pad. The user enters the letters corresponding to their PIN in the box.

Description of System (Plaz)

Original Off the Grid

We were inspired by the "Off the Grid" system from the Gibson Research Corporation.[1] The "Off the Grid" proposal is designed to allow users to use a personal printed paper grid to encipher the domain name of the website they are currently on into a string of psudeo-random characters.

The Off the Grid system works entirely on the user's side. Websites do not need to do anything to support Off the Grid.

To use Off the Grid, the user first generates a grid from a grid-providing website such as https://www.grc.com/offthegrid.htm. This website generates a grid using client-side scripting (ie. JavaScript) to generate the grid on the user's machine. The user then prints the grid onto a sheet of letter paper. At this point the Grid is offline and thus impossible to access by malware. As an alternative, there is at least one application for Android which produces and stores a grid; however, the grid is now accessible to malware on the Android phone which is able to defeat the inter-process sandboxing.

The grid that is generated is a Latin Square. A Latin Square is an $n \times n$ array filled with n different symbols, each occurring exactly once in each row and exactly once in each column.[2] The most famous Latin Square is the popular puzzle game Sudoku. (Note however, that we do not divide up the grid into 9 smaller 3x3 mini-squares in which each symbol must be unique). For example, here is a 11x11 Latin Square with 11 alphabetic characters:

g	e	a	m	o	n	z	k	i	r	c
k	a	n	c	m	z	o	r	g	i	e
n	k	c	z	a	m	r	g	o	e	i
z	o	i	a	n	g	e	c	r	k	m
m	r	z	n	g	a	k	i	e	c	o
a	g	e	i	z	r	n	o	c	m	k
r	c	g	k	e	i	m	n	z	o	a
e	n	k	g	i	o	c	z	m	a	r
i	m	o	e	r	c	g	a	k	n	k
c	i	r	o	k	e	a	m	n	z	g
o	z	m	r	c	k	i	e	a	g	n

Figure 2

Once the user has a grid, they use the grid to create or change the password for each website. The Off the Grid specification has a number of variants, but we will use the base variant described on the GRC website.

In the Off the Grid specification, the user traces the name of the website twice to provide additional entropy. In the start of the first phase, the user always starts along the first row of the grid.

---

[1] https://www.grc.com/offthegrid.htm and associated pages. Is still marked as "Work in Progress;" Retrieved 12/2/2012

[2] http://en.wikipedia.org/wiki/Latin_square

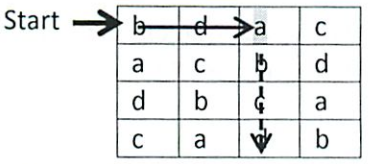


Figure 3

The user then traces out the first 6 characters of the domain name. 6 characters was chosen by the author to provide a 12 character password, which the author chose to balance ease of use with entropy. Again, a user may choose their own scheme. The user alternates between looking horizontally and vertically.

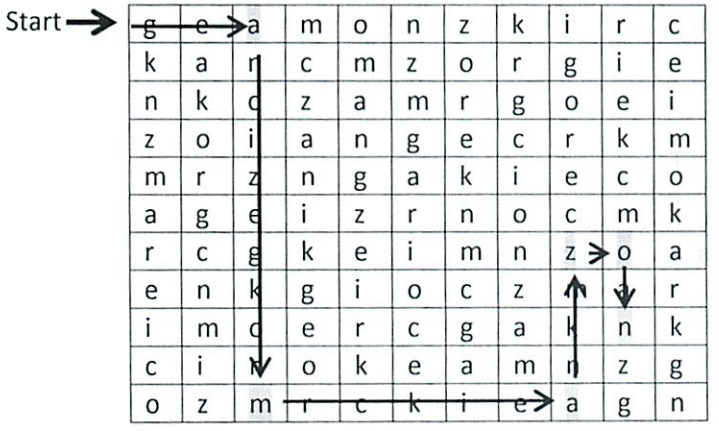


Figure 4

In the second phase, the user starts at the character that they ended with at the end of Phase 1. The user then two more characters from the grid in the same direction of travel. The user then appends those two characters to their password.

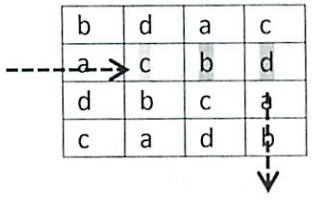


Figure 5 The user arrives at c traveling to the right. The user appends the next two characters "bd" to their password, and then continues up/down from the last character they read "d".

The user wraps around if their characters go off the grid.

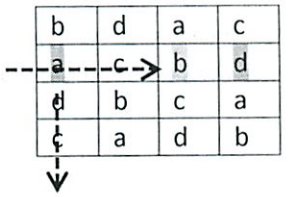
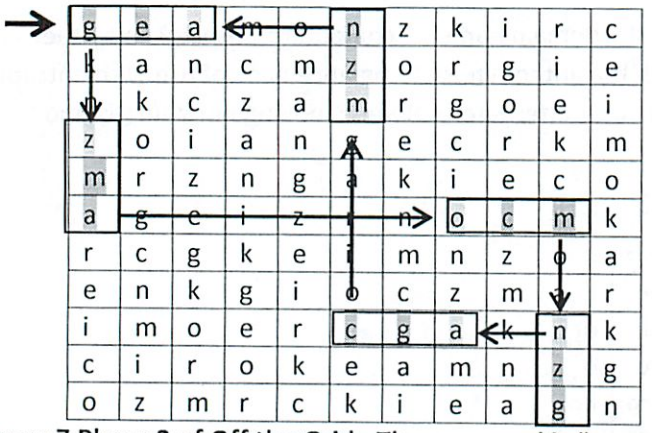


Figure 6 The user arrives at b traveling to the right. The user appends the next two characters to their password, wrapping around if they go off the edge of the grid. Here those characters are "da". The user then continues up/down from the last character "a".

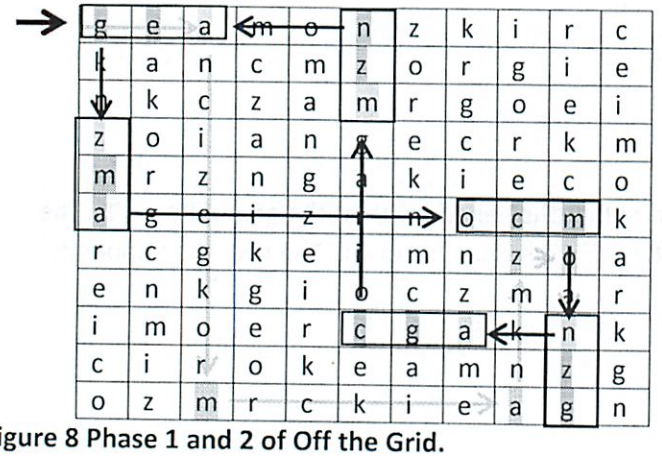
For example here is Phase 2 of our Amazon example.



g	e	a	m	o	n	z	k	i	r	c
l	a	n	c	m	z	o	r	g	i	e
v	k	c	z	a	m	r	g	o	e	i
z	o	i	a	n	g	e	c	r	k	m
m	r	z	n	g	a	k	i	e	c	o
a	g	e	i	z	d	n	o	c	m	k
r	c	g	k	e	i	m	n	z	o	a
e	n	k	g	i	o	c	z	m	a	r
i	m	o	e	r	c	g	a	k	n	k
c	i	r	o	k	e	a	m	n	z	g
o	z	m	r	c	k	i	e	a	g	n

Figure 7 Phase 2 of Off the Grid. The password is "gaznegmacmzg"

Here are Phase 1 and Phase 2.



g	e	a	m	o	n	z	k	i	r	c
l	a	n	c	m	z	o	r	g	i	e
v	k	c	z	a	m	r	g	o	e	i
z	o	i	a	n	g	e	c	r	k	m
m	r	z	n	g	a	k	i	e	c	o
a	g	e	i	z	d	n	o	c	m	k
r	c	g	k	e	i	m	n	z	o	a
e	n	k	g	i	o	c	z	m	a	r
i	m	o	e	r	c	g	a	k	n	k
c	i	r	o	k	e	a	m	n	z	g
o	z	m	r	c	k	i	e	a	g	n

Figure 8 Phase 1 and 2 of Off the Grid.

To log in, the user retraces exactly the same steps as when creating a password. This means the password is exactly the same for each domain. This is an obvious requirement for a system designed to fit within the existing password infrastructure. However, we wanted to explore ideas in which the user does not enter the same password each time.

Tracer: Trace the Grid

We wanted to design a system similar to the Off the Grid system, but where the password the user transmits over the network is different each time. With this system, the website presents the user with a grid and the user enters only a deviation of their password.

When the user creates an account, he provides his or her password to the webserver. The user may use characters from the lower case Latin alphabet [a...z].

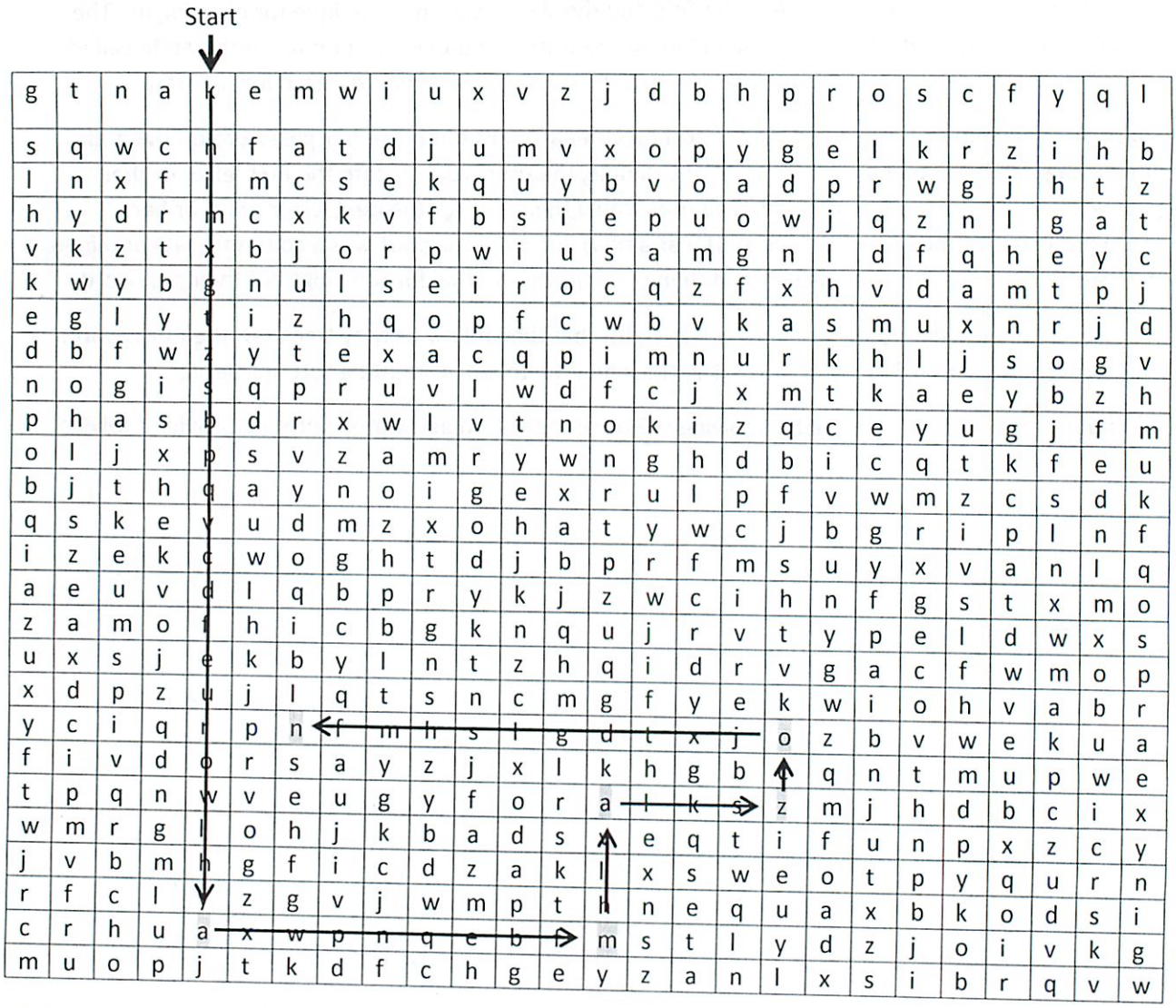
When the user logs in, the server randomly generates a 26x26 Latin square with the characters [a...z] called the *Grid*. The server also randomly selects a start row or column called the *start location*. The server transmits this Grid to the user. The Grid and the start location are unique for each log in. The server stores the Grid and start location in temporary state and provides a pointer to this state called the *token* to the user. The user's browser returns the token to the server on each log in attempt.

These are transmitted to the user. The user then visually traces out his or her password on the grid, alternating between rows and columns. For example, the user would locate the first letter of their password on the start row or column. The user would then look for the next letter of his or her password in either the column (if the start was a row) or row (if the start was a column) that contained the user's first character. The user would then continue alternating for the length of their password.

The user enters the directions (up, down, left, right) that they follow as they trace out their password. This is called the *trace* of the password. The trace and the token are sent back to the server.

The server verifies that the trace by applying the trace to the grid associated with the provided token.

Example: entering the password Amazon with the 5th column as the start row/column. The grid as well as the start row/column are randomly generated be the server for each log in.



The resulting trace would be: Down, Right, Up, Right, Up, Left.

Figure 9 A trace of the password "amazon"

Modified Tracer

We also explored a modified version of this system designed to increase usability. This system uses a 13x13 grid, instead of a 26x26 grid to make it easier for users to visually scan the grid.

In addition, we no longer generate a Latin Square. Instead, we first randomly distribute the user's password in an empty grid. We first randomly select either a row or a column from our 26 choices. We then place the first letter somewhere in that row or column. For this example, say we select the 3rd column to start with. We then place the "a" somewhere in this first column. We then place the second

letter "m" in the row in which we have placed the first letter. We continue this scheme until the password has been placed.

For example:

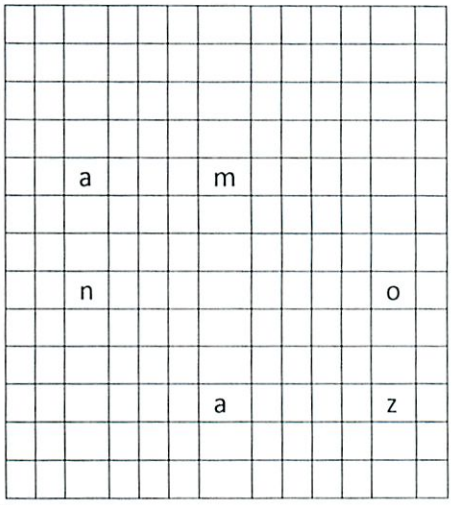


Figure 10

We then randomly fill in the remaining letters on the grid from the set of 26 lower case letters. We make sure each row and column only contains each letter only once by backtracking. For each spot we first start with the entire set [a...z]. We then remove all letters that are currently in the same row and column that we are in. We then randomly select a character from the remaining set.

This is not a Latin Square because we have a [13x13] Grid, but 26 possible characters.

Probabilistic Analysis (Miguel)

Original Off the Grid

Tracer: Trace the Grid

Modified Tracer

Math from breaking down the Latin Square.

Other Factors (Plaz)

We evaluate each system according to the criteria set out in The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes.[3]

[3] http://css.csail.mit.edu/6.858/2012/readings/passwords.pdf

Original Off the Grid

Tracer: Trace the Grid

Modified Tracer

Usability (Plaz)

The simpler a system is, the more it will be used.

6.813 UI class

Original Off the Grid

Tracer: Trace the Grid

Modified Tracer

Code (Jwang)

Anything we want to write here?

How do we generate a Latin Square?

Conclusion (Jwang)

6.858
Meeting

12/2

Will write up tonight → Miguel
started research

No Jwang

CrossPassword or Tracer?

What does code say?
Consistent w/ my defn?
We'll see what is essential...

his placement seems weird - ok?

6.858
Writeup

12/2
3:30P

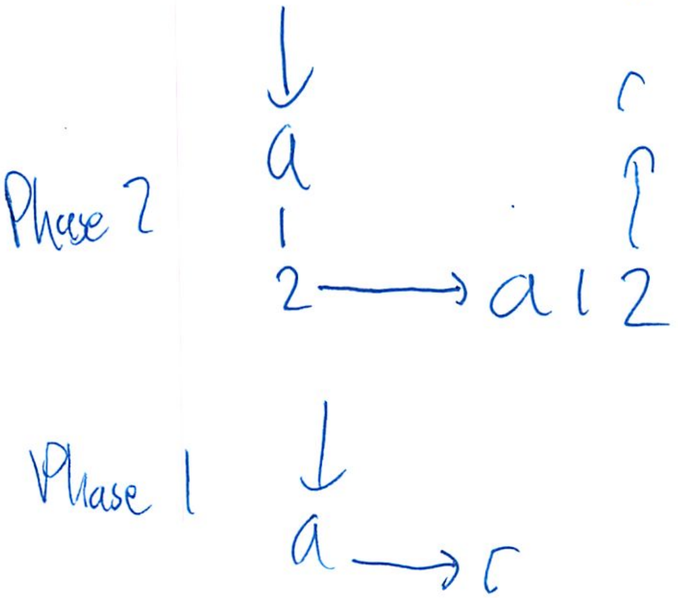
Why did Juang put orthogonal Latin sq

That is when have 2 symbols that must be unique in each...

Repeating ~~password~~ character

aardvark

How would GRC support



not skip

Though he never saves

(2)

Browser add on that solves

¡ underline capitalize or italics key terms ¡ each time ¡

The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes*

Joseph Bonneau
University of Cambridge
Cambridge, UK
jcb82@cl.cam.ac.uk

Cormac Herley
Microsoft Research
Redmond, WA, USA
cormac@microsoft.com

Paul C. van Oorschot
Carleton University
Ottawa, ON, Canada
paulv@scs.carleton.ca

Frank Stajano†
University of Cambridge
Cambridge, UK
frank.stajano@cl.cam.ac.uk

***Abstract*—We evaluate two decades of proposals to replace text passwords for general-purpose user authentication on the web using a broad set of twenty-five usability, deployability and security benefits that an ideal scheme might provide. The scope of proposals we survey is also extensive, including password management software, federated login protocols, graphical password schemes, cognitive authentication schemes, one-time passwords, hardware tokens, phone-aided schemes and biometrics. Our comprehensive approach leads to key insights about the difficulty of replacing passwords. Not only does no known scheme come close to providing all desired benefits: none even retains the full set of benefits that legacy passwords already provide. In particular, there is a wide range from schemes offering minor security benefits beyond legacy passwords, to those offering significant security benefits in return for being more costly to deploy or more difficult to use. We conclude that many academic proposals have failed to gain traction because researchers rarely consider a sufficiently wide range of real-world constraints. Beyond our analysis of current schemes, our framework provides an evaluation methodology and benchmark for future web authentication proposals.**

***Keywords*-authentication; computer security; human computer interaction; security and usability; deployability; economics; software engineering.**

I. Introduction

The continued domination of passwords over all other methods of end-user authentication is a major embarrassment to security researchers. As web technology moves ahead by leaps and bounds in other areas, passwords stubbornly survive and reproduce with every new web site. Extensive discussions of alternative authentication schemes have produced no definitive answers.

Over forty years of research have demonstrated that passwords are plagued by security problems [2] and openly hated by users [3]. We believe that, to make progress, the community must better systematize the knowledge that we have regarding both passwords and their alternatives [4]. However, among other challenges, unbiased evaluation of password replacement schemes is complicated by the diverse interests of various communities. In our experience, security experts focus more on security but less on usability and practical issues related to deployment; biometrics experts focus on analysis of false negatives and naturally-occurring false positives rather than on attacks by an intelligent, adaptive adversary; usability experts tend to be optimistic about security; and originators of a scheme, whatever their background, downplay or ignore benefits that their scheme doesn't attempt to provide, thus overlooking dimensions on which it fares poorly. As proponents assert the superiority of their schemes, their objective functions are often not explicitly stated and differ substantially from those of potential adopters. Targeting different authentication problems using different criteria, some address very specific environments and narrow scenarios; others silently seek generic solutions that fit all environments at once, assuming a single choice is mandatory. As such, consensus is unlikely.

These and other factors have contributed to a long-standing lack of progress on how best to evaluate and compare authentication proposals intended for practical use. In response, we propose a standard benchmark and framework allowing schemes to be rated across a common, broad spectrum of criteria chosen objectively for relevance in wide-ranging scenarios, without hidden agenda.[1] We suggest and define 25 properties framed as a diverse set of benefits, and a methodology for comparative evaluation, demonstrated and tested by rating 35 password-replacement schemes on the same criteria, as summarized in a carefully constructed comparative table.

Both the rating criteria and their definitions were iteratively refined over the evaluation of these schemes. Discussion of evaluation details for passwords and nine representative alternatives is provided herein to demonstrate the process, and to provide evidence that the list of benefits suffices to illuminate the strengths and weaknesses of a wide universe of schemes. Though not cast in stone, we believe that the list of benefits and their specific definitions provide an excellent basis from which to work; the framework and

*An extended version of this paper is available as a University of Cambridge technical report [1].

†Frank Stajano was the lead author who conceived the project and assembled the team. All authors contributed equally thereafter.

[1]The present authors contributed to the definition of the following schemes: URRSA [5], MP-Auth [6], PCCP [7] and Pico [8]. We invite readers to verify that we have rated them impartially.

In *Proc. IEEE Symp. on Security and Privacy 2012* ("Oakland 2012").

evaluation process that we define are independent of them, although our comparative results naturally are not. From our analysis and comparative summary table, we look for clues to help explain why passwords remain so dominant, despite frequent claims of superior alternatives.

In the past decade our community has recognized a tension between security and usability: it is generally easy to provide more of one by offering less of the other. But the situation is much more complex than simply a linear trade-off: we seek to capture the multi-faceted, rather than one-dimensional, nature of both usability and security in our benefits. We further suggest that "deployability", for lack of a better word, is an important third dimension that deserves consideration. We choose to examine all three explicitly, complementing earlier comparative surveys (e.g., [9]–[11]).

Our usability-deployability-security ("UDS") evaluation framework and process may be referred to as *semi-structured evaluation of user authentication schemes*. We take inspiration from inspection methods for evaluating user interface design, including *feature inspections* and Nielsen's *heuristic analysis* based on usability principles [12].

Each co-author acted as a domain expert, familiar with both the rating framework and a subset of the schemes. For each scheme rated, the evaluation process involved one co-author studying the scheme and rating it on the defined benefits; additional co-authors reviewing each rating score; and iteratively refining the ratings as necessary through discussion, as noted in Section V-D.

Our focus is *user authentication on the web*, specifically from unsupervised end-user client devices (e.g., a personal computer) to remote verifiers. Some schemes examined involve mobile phones as auxiliary devices, but logging in directly from such constrained devices, which involves different usability challenges among other things, is not a main focus. Our present work does not directly examine schemes designed exclusively for machine-to-machine authentication, e.g., cryptographic protocols or infrastructure such as client public-key certificates. Many of the schemes we examine, however, are the technologies proposed for the human-to-machine component that may precede machine-to-machine authentication. Our choice of web authentication as target application also has significant implications for specific schemes, as noted in our results.

II. Benefits

The benefits we consider encompass three categories: usability, deployability and security, the latter including privacy aspects. The benefits in our list have been refined to a set we believe highlights important evaluation dimensions, with an eye to limiting overlap between benefits.

Throughout the paper, for brevity and consistency, each benefit is referred to with an italicized mnemonic title. This title should not be interpreted too literally; refer instead to our actual definitions below, which are informally worded to aid use. Each scheme is rated as either offering or not offering the benefit; if a scheme *almost* offers the benefit, but not quite, we indicate this with the *Quasi-* prefix. Section V-D discusses pros and cons of finer-grained scoring.

Sometimes a particular benefit (e.g., *Resilient-to-Theft*) just doesn't apply to a particular scheme (e.g., there is nothing physical to steal in a scheme where the user must memorize a secret squiggle). To simplify analysis, instead of introducing a "not applicable" value, we rate the scheme as offering the benefit—in the sense that nothing can go wrong, for that scheme, with respect to the corresponding problem.

When rating password-related schemes we assume that implementers use best practice such as salting and hashing (even though we know they often don't [13]), because we assess what the scheme's design can potentially offer: a poor implementation could otherwise kill any scheme. On the other hand, we assume that ordinary users won't necessarily follow the often unreasonably inconvenient directives of security engineers, such as never recycling passwords, or using randomly-generated ones.

A. Usability benefits

U1 *Memorywise-Effortless*: Users of the scheme do not have to remember *any* secrets at all. We grant a *Quasi-Memorywise-Effortless* if users have to remember *one* secret for everything (as opposed to one per verifier).

U2 *Scalable-for-Users*: Using the scheme for hundreds of accounts does not increase the burden on the user. As the mnemonic suggests, we mean "scalable" only from the user's perspective, looking at the cognitive load, not from a system deployment perspective, looking at allocation of technical resources.

U3 *Nothing-to-Carry*: Users do not need to carry an additional physical object (electronic device, mechanical key, piece of paper) to use the scheme. *Quasi-Nothing-to-Carry* is awarded if the object is one that they'd carry everywhere all the time anyway, such as their mobile phone, but not if it's their computer (including tablets).

U4 *Physically-Effortless*: The authentication process does not require physical (as opposed to cognitive) user effort beyond, say, pressing a button. Schemes that don't offer this benefit include those that require typing, scribbling or performing a set of motions. We grant *Quasi-Physically-Effortless* if the user's effort is limited to speaking, on the basis that even illiterate people find that natural to do.

U5 *Easy-to-Learn*: Users who don't know the scheme can figure it out and learn it without too much trouble, and then easily recall how to use it.

U6 *Efficient-to-Use*: The time the user must spend for each authentication is acceptably short. The time

required for setting up a new association with a verifier, although possibly longer than that for authentication, is also reasonable.

U7 *Infrequent-Errors*: The task that users must perform to log in usually succeeds when performed by a legitimate and honest user. In other words, the scheme isn't so hard to use or unreliable that genuine users are routinely rejected.[2]

U8 *Easy-Recovery-from-Loss*: A user can conveniently regain the ability to authenticate if the token is lost or the credentials forgotten. This combines usability aspects such as: low latency before restored ability; low user inconvenience in recovery (e.g., no requirement for physically standing in line); and assurance that recovery will be possible, for example via built-in backups or secondary recovery schemes. If recovery requires some form of re-enrollment, this benefit rates its convenience.

B. Deployability benefits

D1 *Accessible*: Users who can use passwords[3] are not prevented from using the scheme by disabilities or other physical (not cognitive) conditions.

D2 *Negligible-Cost-per-User*: The total cost per user of the scheme, adding up the costs at both the prover's end (any devices required) and the verifier's end (any share of the equipment and software required), is negligible. The scheme is plausible for startups with no per-user revenue.

D3 *Server-Compatible*: At the verifier's end, the scheme is compatible with text-based passwords. Providers don't have to change their existing authentication setup to support the scheme.

D4 *Browser-Compatible*: Users don't have to change their client to support the scheme and can expect the scheme to work when using other machines with an up-to-date, standards-compliant web browser and no additional software. In 2012, this would mean an HTML5-compliant browser with JavaScript enabled. Schemes fail to provide this benefit if they require the installation of plugins or any kind of software whose installation requires administrative rights. Schemes offer *Quasi-Browser-Compatible* if they rely on non-standard but very common plugins, e.g., Flash.

D5 *Mature*: The scheme has been implemented and deployed on a large scale for actual authentication purposes beyond research. Indicators to consider for granting the full benefit may also include whether the scheme has undergone user testing, whether the standards community has published related documents, whether open-source projects implementing the scheme exist, whether anyone other than the implementers has adopted the scheme, the amount of literature on the scheme and so forth.

D6 *Non-Proprietary*: Anyone can implement or use the scheme for any purpose without having to pay royalties to anyone else. The relevant techniques are generally known, published openly and not protected by patents or trade secrets.

C. Security benefits

S1 *Resilient-to-Physical-Observation*: An attacker cannot impersonate a user after observing them authenticate one or more times. We grant *Quasi-Resilient-to-Physical-Observation* if the scheme could be broken only by repeating the observation more than, say, 10–20 times. Attacks include shoulder surfing, filming the keyboard, recording keystroke sounds, or thermal imaging of keypad.

S2 *Resilient-to-Targeted-Impersonation*: It is not possible for an acquaintance (or skilled investigator) to impersonate a specific user by exploiting knowledge of personal details (birth date, names of relatives etc.). Personal knowledge questions are the canonical scheme that fails on this point.

S3 *Resilient-to-Throttled-Guessing*: An attacker whose rate of guessing is constrained by the verifier cannot successfully guess the secrets of a significant fraction of users. The verifier-imposed constraint might be enforced by an online server, a tamper-resistant chip or any other mechanism capable of throttling repeated requests. To give a quantitative example, we might grant this benefit if an attacker constrained to, say, 10 guesses per account per day, could compromise at most 1% of accounts in a year. Lack of this benefit is meant to penalize schemes in which it is frequent for user-chosen secrets to be selected from a small and well-known subset (low min-entropy [14]).

S4 *Resilient-to-Unthrottled-Guessing*: An attacker whose rate of guessing is constrained only by available computing resources cannot successfully guess the secrets of a significant fraction of users. We might for example grant this benefit if an attacker capable of attempting up to 2^{40} or even 2^{64} guesses per account could still only reach

[2] We could view this benefit as "low false reject rate". In many cases the scheme designer could make the false reject rate lower by making the false accept rate higher. If this is taken to an extreme we count it as cheating, and penalize it through a low score in some of the security-related benefits.

[3] Ideally a scheme would be usable by everyone, regardless of disabilities like zero-vision (blindness) or low motor control. However, for any given scheme, it is always possible to identify a disability or physical condition that would exclude a category of people and then no scheme would be granted this benefit. We therefore choose to award the benefit to schemes that do at least as well as the incumbent that is de facto accepted today, despite the fact that it too isn't perfect. An alternative to this text password baseline could be to base the metric on the ability to serve a defined percentage of the population of potential users.

fewer than 1% of accounts. Lack of this benefit is meant to penalize schemes where the space of credentials is not large enough to withstand brute force search (including dictionary attacks, rainbow tables and related brute force methods smarter than raw exhaustive search, if credentials are user-chosen secrets).

S5 *Resilient-to-Internal-Observation*: An attacker cannot impersonate a user by intercepting the user's input from inside the user's device (e.g., by key-logging malware) or eavesdropping on the cleartext communication between prover and verifier (we assume that the attacker can also defeat TLS if it is used, perhaps through the CA). As with *Resilient-to-Physical-Observation* above, we grant *Quasi-Resilient-to-Internal-Observation* if the scheme could be broken only by intercepting input or eavesdropping cleartext more than, say, 10–20 times. This penalizes schemes that are not replay-resistant, whether because they send a static response or because their dynamic response countermeasure can be cracked with a few observations. This benefit assumes that general-purpose devices like software-updatable personal computers and mobile phones may contain malware, but that hardware devices dedicated exclusively to the scheme can be made malware-free. We grant *Quasi-Resilient-to-Internal-Observation* to two-factor schemes where both factors must be malware-infected for the attack to work. If infecting only one factor breaks the scheme, we don't grant the benefit.

S6 *Resilient-to-Leaks-from-Other-Verifiers*: Nothing that a verifier could possibly leak can help an attacker impersonate the user to another verifier. This penalizes schemes where insider fraud at one provider, or a successful attack on one back-end, endangers the user's accounts at other sites.

S7 *Resilient-to-Phishing*: An attacker who simulates a valid verifier (including by DNS manipulation) cannot collect credentials that can later be used to impersonate the user to the actual verifier. This penalizes schemes allowing phishers to get victims to authenticate to lookalike sites and later use the harvested credentials against the genuine sites. It is not meant to penalize schemes vulnerable to more sophisticated real-time man-in-the-middle or relay attacks, in which the attackers have one connection to the victim prover (pretending to be the verifier) and simultaneously another connection to the victim verifier (pretending to be the prover).

S8 *Resilient-to-Theft*: If the scheme uses a physical object for authentication, the object cannot be used for authentication by another person who gains possession of it. We still grant *Quasi-Resilient-to-Theft* if the protection is achieved with the modest strength of a PIN, even if attempts are not rate-controlled, because the attack doesn't easily scale to many victims.

S9 *No-Trusted-Third-Party*: The scheme does not rely on a trusted third party (other than the prover and the verifier) who could, upon being attacked or otherwise becoming untrustworthy, compromise the prover's security or privacy.

S10 *Requiring-Explicit-Consent*: The authentication process cannot be started without the explicit consent of the user. This is both a security and a privacy feature (a rogue wireless RFID-based credit card reader embedded in a sofa might charge a card without user knowledge or consent).

S11 *Unlinkable*: Colluding verifiers cannot determine, from the authenticator alone, whether the same user is authenticating to both. This is a privacy feature. To rate this benefit we disregard linkability introduced by other mechanisms (same user ID, same IP address, etc).

We emphasize that it would be simple-minded to rank competing schemes simply by counting how many benefits each offers. Clearly some benefits deserve more weight than others—but which ones? *Scalable-for-Users*, for example, is a heavy-weight benefit if the goal is to adopt a single scheme as a universal replacement; it is less important if one is seeking a password alternative for only a single account. Providing appropriate weights thus depends strongly on the specific goal for which the schemes are being compared, which is one of the reasons we don't offer any.

Having said that, readers wanting to use weights might use our framework as follows. First, examine and score each individual scheme on each benefit; next, compare (groups of) competing schemes to identify precisely which benefits each offers over the other; finally, with weights that take into account the relative importance of the benefits, determine an overall ranking by rating scheme i as $S_i = \sum_j W_j \cdot b_{i,j}$. Weights W_j are constants across all schemes in a particular comparison exercise, and $b_{i,j} \in [0, 1]$ is the real-valued benefit rating for scheme i on benefit j. For different solution environments (scenarios k), the relative importance of benefits will differ, with weights W_j replaced by $W_j^{(k)}$.

In this paper we choose a more qualitative approach: we do not suggest any weights $W_j^{(k)}$ and the $b_{i,j}$ ratings we assign are not continuous but coarsely quantized. In Section V-D we discuss why. In our experience, *"the journey* (the rating exercise) *is the reward"*: the important technical insights we gained about schemes by discussing whether our ratings were fair and consistent were worth much more to us than the actual scores produced. As a take-home message for the value of this exercise, bringing a team of experts to

a shared understanding of the relevant technical issues is much more valuable than ranking the schemes linearly or reaching unanimous agreement over scoring.

III. Evaluating legacy passwords

We expect that the reader is familiar with text passwords and their shortcomings, so evaluating them is good exercise for our framework. It's also useful to have a baseline standard to refer to. While we consider "legacy passwords" as a single scheme, surveys of password deployment on the web have found substantial variation in implemention. A study of 150 sites in 2010 [13], for example, found a unique set of design choices at nearly every site. Other studies have focused on implementations of cookie semantics [15], password composition policies [16], or use of TLS to protect passwords [17]. Every study has found both considerable inconsistency and frequent serious implementation errors in practical deployments on the web.

We remind readers of our Section II assumption of best practice by implementers—thus in our ratings we do not hold against passwords the many weak implementations that their widespread deployment includes, unless due to inherent weaknesses; while on the other hand, our ratings of passwords and other schemes do assume that poor user behavior is an inherent aspect of fielded systems.

The difficulty of guessing passwords was studied over three decades ago [2] with researchers able to guess over 75% of users' passwords; follow-up studies over the years have consistently compromised a substantial fraction of accounts with dictionary attacks. A survey [3] of corporate password users found them flustered by password requirements and coping by writing passwords down on post-it notes. On the web, users are typically overwhelmed by the number of passwords they have registered. One study [18] found most users have many accounts for which they've forgotten their passwords and even accounts they can't remember registering. Another [19] used a browser extension to observe thousands of users' password habits, finding on average 25 accounts and 6 unique passwords per user.

Thus, passwords, as a purely memory-based scheme, clearly aren't *Memorywise-Effortless* or *Scalable-for-Users* as they must be remembered and chosen for each site. While they are *Nothing-to-Carry*, they aren't *Physically-Effortless* as they must be typed. Usability is otherwise good, as passwords are de facto *Easy-to-Learn* due to years of user experience and *Efficient-to-Use* as most users type only a few characters, though typos downgrade passwords to *Quasi-Infrequent-Errors*. Passwords can be easily reset, giving them *Easy-Recovery-from-Loss*.

Their highest scores are in deployability, where they receive full credit for every benefit—in part because many of our criteria are defined based on passwords. For example, passwords are *Accessible* because we defined the benefit with respect to them and accommodations already exist for most groups due to the importance of passwords. Passwords are *Negligible-Cost-per-User* due to their simplicity, and are *Server-Compatible* and *Browser-Compatible* due to their incumbent status. Passwords are *Mature* and *Non-Proprietary*, with turnkey packages implementing password authentication for many popular web development platforms, albeit not well-standardized despite their ubiquity.

Passwords score relatively poorly on security. They aren't *Resilient-to-Physical-Observation* because even if typed quickly they can be automatically recovered from high-quality video of the keyboard [20]. Perhaps generously, we rate passwords as *Quasi-Resilient-to-Targeted-Impersonation* in the absence of user studies establishing acquaintances' ability to guess passwords, though many users undermine this by keeping passwords written down in plain sight [3]. Similarly, users' well-established poor track record in selection means passwords are neither *Resilient-to-Throttled-Guessing* nor *Resilient-to-Unthrottled-Guessing*.

As static tokens, passwords aren't *Resilient-to-Internal-Observation*. The fact that users reuse them across sites means they also aren't *Resilient-to-Leaks-from-Other-Verifiers*, as even a properly salted and strengthened hash function [21] can't protect many passwords from dedicated cracking software. (Up to 50% of websites don't appear to hash passwords at all [13].) Passwords aren't *Resilient-to-Phishing* as phishing remains an open problem in practice.

Finally, their simplicity facilitates several security benefits. They are *Resilient-to-Theft* as they require no hardware. There is *No-Trusted-Third-Party*; having to type makes them *Requiring-Explicit-Consent*; and, assuming that sites add salt independently, even weak passwords are *Unlinkable*.

IV. Sample evaluation of replacement schemes

We now use our criteria to evaluate a representative sample of proposed password replacement schemes. Table I visually summarizes these and others we explored. Due to space constraints, we only explain in detail our ratings for at most one representative scheme per category (e.g. federated login schemes, graphical passwords, hardware tokens, etc.). Evaluation details for all other schemes in the table are provided in a companion technical report [1].

We introduce categories to highlight general trends, but stress that any scheme must be rated individually. Contrary to what the table layout suggests, schemes are not uniquely partitioned by the categories; several schemes belong to multiple categories, and different groupings of the schemes are possible with these same categories. For example, GrIDsure is both cognitive and graphical; and, though several of the schemes we examine use some form of underlying "one-time-passwords", we did not group them into a common category and indeed have no formal category of that name.

We emphasize that, in selecting a particular scheme for inclusion in the table or for discussion as a category representative, we do not necessarily endorse it as better than

alternatives—merely that it is reasonably representative, or illuminates in some way what the category can achieve.

A. Encrypted password managers: Mozilla Firefox

The Firefox web browser [22] automatically offers to remember passwords entered into web pages, optionally encrypting them with a master password. (Our rating assumes that this option is used; use without the password has different properties.) It then pre-fills the username and password fields when the user revisits the same site. With its Sync facility the passwords can be stored, encrypted, in the cloud. After a once-per-machine authentication ritual, they are updated automatically on all designated machines.

This scheme is *Quasi-Memorywise-Effortless* (because of the master password) and *Scalable-for-Users*: it can remember arbitrarily many passwords. Without Sync, the solution would have required carrying a specific computer; with Sync, the passwords can be accessed from any of the user's computers. However it's not more than *Quasi-Nothing-to-Carry* because a travelling user will have to carry at least a smartphone: it would be quite insecure to sync one's passwords with a browser found in a cybercafé. It is *Quasi-Physically-Effortless*, as no typing is required during authentication except for the master password once per session, and *Easy-to-Learn*. It is *Efficient-to-Use* (much more so than what it replaces) and has *Infrequent-Errors* (hardly any, except when entering the master password). It does not have *Easy-Recovery-from-Loss*: losing the master password is catastrophic.

The scheme is backwards-compatible by design and thus scores quite highly on deployability: it fully provides all the deployability benefits except for *Browser-Compatible*, unavoidably because it requires a specific browser.

It is *Quasi-Resilient-to-Physical-Observation* and *Quasi-Resilient-to-Targeted-Impersonation* because an attacker could still target the infrequently-typed master password (but would also need access to the browser). It is not *Resilient-to-Throttled-Guessing* nor *Resilient-to-Unthrottled-Guessing*: even if the master password is safe from such attacks, the original web passwords remain as vulnerable as before.[4] It is not *Resilient-to-Internal-Observation* because, even if TLS is used, it's replayable static passwords that flow in the tunnel and malware could also capture the master password. It's not *Resilient-to-Leaks-from-Other-Verifiers*, because what happens at the back-end is the same as with passwords. It's *Resilient-to-Phishing* because we assume that sites follow best practice, which includes using TLS for the login page. It is *Resilient-to-Theft*, at least under our assumption that a master password is being used. It offers *No-Trusted-Third-Party* because the Sync data is pre-encrypted locally before being stored on Mozilla's servers. It offers *Requiring-Explicit-Consent* because it pre-fills the username and password fields but the user still has to press enter to submit. Finally, it is as *Unlinkable* as passwords.

[4]Security-conscious users might adopt truly random unguessable passwords, as they need no longer remember them, but most users won't. If the scheme pre-generated random passwords it would score more highly here, disregarding pre-existing passwords. Similarly, for *Resilient-to-Leaks-from-Other-Verifiers* below, this scheme makes it easier for careful users to use a different password for every site; if it forced this behaviour (vs. just allowing it), it would get a higher score on this particular benefit.

B. Proxy-based: URRSA

Proxy-based schemes place a man-in-the-middle between the user's machine and the server. One reason for doing so, employed by Impostor [23] and URRSA [5] is to enable secure logins despite malware-infected clients.

URRSA has users authenticate to the end server using one-time codes carried on a sheet of paper. At registration the user enters the password, P_j, for each account, j, to be visited; this is encrypted at the proxy with thirty different keys, K_i, giving $C_i = E_{K_i}(P_j)$. The C_i act as one-time codes which the user prints and carries. The codes are generally 8-10 characters long; thirty codes for each of six accounts fit on a two-sided sheet. The keys, but not the passwords, are stored at the proxy. At login the user visits the proxy, indicates which site is desired, and is asked for the next unused code. When he enters the code it is decrypted and passed to the end login server: $E_{K_i}^{-1}(C_i) = P_j$. The proxy never authenticates the user, it merely decrypts with an agreed-upon key, the code delivered by the user.

Since it requires carrying one-time codes URRSA is *Memorywise-Effortless*, but not *Scalable-for-Users* or *Nothing-to-Carry*. It is not *Physically-Effortless* but is *Easy-to-Learn*. In common with all of the schemes that involve transcribing codes from a device or sheet it is not *Efficient-to-Use*. However, we do consider it to have *Quasi-Infrequent-Errors*, since the codes are generally 8-10 characters. It does not have *Easy-Recovery-from-Loss*: a revocation procedure is required if the code sheet is lost or stolen. Since no passwords are stored at the proxy the entire registration must be repeated if this happens.

In common with other paper token schemes it is not *Accessible*. URRSA has *Negligible-Cost-per-User*. Rather than have a user change browser settings, URRSA relies on a link-translating proxy that intermediates traffic between the user and the server; this translation is not flawless and some functionality may fail on complex sites, thus we consider it only *Quasi-Server-Compatible*. It is, however, *Browser-Compatible*. It is neither *Mature* nor *Non-Proprietary*.

In common with other one-time code schemes it is not *Resilient-to-Physical-Observation*, since a camera might capture all of the codes on the sheet. Since it merely inserts a proxy it inherits many security weaknesses from the legacy password system it serves: it is *Quasi-Resilient-to-Targeted-Impersonation* and is not *Resilient-to-Throttled-Guessing* or *Resilient-to-Unthrottled-Guessing*. It is *Quasi-Resilient-to-Internal-Observation* as observing the client during authentication does not allow passwords to be captured, but breaking

the proxy-to-server TLS connection does. It inherits from passwords the fact that it is not *Resilient-to-Leaks-from-Other-Verifiers*, but the fact that it is *Resilient-to-Phishing* from other one-time schemes. It is not *Resilient-to-Theft* nor *No-Trusted-Third-Party*: the proxy must be trusted. It offers *Requiring-Explicit-Consent* and is *Unlinkable*.

C. Federated Single Sign-On: OpenID

Federated single sign-on enables web sites to authenticate a user by redirecting them to a trusted identity server which attests the users' identity. This has been considered a "holy grail" as it could eliminate the problem of remembering different passwords for different sites. The concept of federated authentication dates at least to the 1978 Needham-Schroeder key agreement protocol [24] which formed the basis for Kerberos [25]. Kerberos has inspired dozens of proposals for federated authentication on the Internet; Pashalidis and Mitchell provided a complete survey [26]. A well-known representative is OpenID,[5] a protocol which allows any web server to act as an "identity provider" [27] to any server desiring authentication (a "relying party"). OpenID has an enthusiastic group of followers both in and out of academia, but it has seen only patchy adoption with many sites willing to act as identity providers but few willing to accept it as relying parties [28].

In evaluating OpenID, we note that in practice identity providers will continue to use text passwords to authenticate users in the forseeable future, although the protocol itself allows passwords to be replaced by a stronger mechanism. Thus, we rate the scheme *Quasi-Memorywise-Effortless* in that most users will still have to remember one master password, but *Scalable-for-Users* as this password can work for multiple sites. OpenID is *Nothing-to-Carry* like passwords and *Quasi-Physically-Effortless* because passwords only need to be typed at the identity provider. Similarly, we rate it *Efficient-to-Use* and *Infrequent-Errors* in that it is either a password authentication or can occur automatically in a browser with cached login cookies for the identity provider. However, OpenID has found that selecting an opaque "identity URL" can be a significant usability challenge without a good interface at the relying party, making the scheme only *Quasi-Easy-to-Learn*. OpenID is *Easy-Recovery-from-Loss*, equivalent to a password reset.

OpenID is favorable from a deployment standpoint, providing all benefits except for *Server-Compatible*, including *Mature* as it has detailed standards and many open-source implementations. We do note however that it requires identity providers yield some control over trust decisions and possibly weaken their own brand [28], a deployment drawback not currently captured in our criteria.

[5]OpenID is often confused with OAuth, a technically unrelated protocol for delegating access to one's accounts to third parties. The recent OpenID Connect proposal merges the two. We consider the OpenID 2.0 standard here, though all current versions score identically in our framework.

Security-wise, OpenID reduces most attacks to only the password authentication between a user and his or her identity provider. This makes it somewhat difficult to rate; we consider it *Quasi-Resilient-to-Throttled-Guessing*, *Quasi-Resilient-to-Unthrottled-Guessing*, *Quasi-Resilient-to-Targeted-Impersonation*, *Quasi-Resilient-to-Physical-Observation* as these attacks are possible but only against the single identity provider (typically cached in a cookie) and not for each login to all verifiers. However, it is not *Resilient-to-Internal-Observation* as malware can either steal persistent login cookies or record the master password. OpenID is also believed to be badly non-*Resilient-to-Phishing* since it involves re-direction to an identity provider from a relying party [29]. OpenID is *Resilient-to-Leaks-from-Other-Verifiers*, as relying parties don't store users passwords. Federated schemes have been criticized on privacy grounds and, while OpenID does enable technically savvy users to operate their own identity provider, we rate OpenID as non-*Unlinkable* and non-*No-Trusted-Third-Party* as the vast majority of users aren't capable of doing so.

D. Graphical passwords: Persuasive Cued Clickpoints (PCCP)

Graphical passwords schemes attempt to leverage natural human ability to remember images, which is believed to exceed memory for text. We consider as a representative PCCP [7] (Persuasive Cued Click-Points), a cued-recall scheme. Users are sequentially presented with five images on each of which they select one point, determining the next image displayed. To log in, all selected points must be correctly re-entered within a defined tolerance. To flatten the password distribution, during password creation a randomly-positioned portal covers a portion of each image; users must select their point from therein (the rest of each image is shaded slightly). Users may hit a "shuffle" button to randomly reposition the portal to a different region—but doing so consumes time, thus persuading otherwise. The portal is absent on regular login. Published security analysis and testing report reasonable usability and improved security over earlier schemes, specifically in terms of resistance to both *hotspots* and *pattern-based* attacks [11].

While not *Memorywise-Effortless*, nor *Scalable-for-Users* due to extra cognitive load for each account password, PCCP offers advantages over text passwords (and other uncued schemes) due to per-account image cues reducing password interference. It is *Easy-to-Learn* (usage and mental models match web passwords, but interface details differ), but only *Quasi-Efficient-to-Use* (login times on the order of 5s to 20s exceed text passwords) and at best *Quasi-Infrequent-Errors*.

PCCP is not *Accessible* (consider blind users) and has *Negligible-Cost-per-User*. It is not *Server-Compatible*; though it might be made so by having a proxy act as intermediary (much as URRSA does). It is *Browser-Compatible*. It is not *Mature*, but apparently *Non-Proprietary*.

PCCP is not *Resilient-to-Physical-Observation* (due to video-camera shoulder surfing), but is *Resilient-to-Targeted-Impersonation* (personal knowledge of a target user does not help attacks). We rate it *Quasi-Resilient-to-Throttled-Guessing* due to portal persuasion increasing password randomness, but note individual users may repeatedly bypass portal recommendations. Although the persuasion is also intended to mitigate offline attacks, we rate it not *Resilient-to-Unthrottled-Guessing* as studies to date have been limited to full password spaces of 2^{43} (which are within reach of offline dictionary attack, especially for users choosing more predictable passwords, assuming verifier-stored hashes are available). It is not *Resilient-to-Internal-Observation* (static passwords are replayable). It is *Resilient-to-Leaks-from-Other-Verifiers* (distinct sites can insist on distinct image sets). PCCP is *Resilient-to-Phishing* per our strict definition of that benefit; to obtain the proper per-user images, a phishing site must interact (e.g., by MITM) with a legitimate server. PCCP matches text passwords on being *Unlinkable*.

E. Cognitive authentication: GrIDsure

Challenge-Response schemes attempt to address the replay attack on passwords by having the user deliver proof that he knows the secret without divulging the secret itself. If memorization and computation were no barrier then the server might challenge the user to return a cryptographic hash of the user's secret combined with a server-selected nonce. However, it is unclear if a scheme within the means of human memory and calculating ability is achievable. We examine the commercial offering GrIDsure (a variant of which is described in a paper [30] by other authors) as representative of the class.

At registration the user is presented with a grid (e.g., 5×5) and selects a pattern, or sequence of cells. There are 25^4 possible length-4 patterns, for example. At login the user is again presented with the grid, but now populated with digits. To authenticate he transcribes the digits in the cells corresponding to his pattern. Since the association of digits to cells is randomized the string typed by the user is different from login to login. Thus he reveals knowledge of his secret without typing the secret itself.

This scheme is similar to passwords in terms of usability and we (perhaps generously) rate it identically in terms of many usability benefits. An exception is that it's only *Quasi-Efficient-to-Use*: unlike passwords, which can often be typed from muscle memory, transcribing digits from the grid cells requires effort and attention and is likely to be slower.

We consider the scheme as not *Accessible* as the two-dimensional layout seems unusable for blind users. The scheme has *Negligible-Cost-per-User*, in terms of technology. It is not *Server-Compatible* but is *Browser-Compatible*. It is not *Mature*. We rate it not *Non-Proprietary*, as the intellectual property status is unknown.

The security properties are, again, similar to passwords in many respects. It is not *Resilient-to-Physical-Observation*, as a camera that captures both the grid and user input quickly learns the secret. It is an improvement on passwords in that it is *Resilient-to-Targeted-Impersonation*: we assume that an attacker is more likely to guess secret strings than secret patterns based on knowledge of the user. However, its small space of choices prevents it from being *Resilient-to-Throttled-Guessing* or *Resilient-to-Unthrottled-Guessing*. In spite of the one-time nature of what the user types the scheme is not *Resilient-to-Internal-Observation*: too many possible patterns are eliminated at each login for the secret to withstand more than three or four observations. It shares the remaining security benefits with passwords.

F. Paper tokens: OTPW

Using paper to store long secrets is the cheapest form of a physical login token. The concept is related to military codebooks used throughout history, but interest in using possession of paper tokens to authenticate humans was spurred in the early 1980's by Lamport's hash-chaining scheme [31], later developed into S/KEY [32]. OTPW is a later refinement, developed by Kuhn in 1998 [33], in which the server stores a larger set of independent hash values, consisting of about 4 kB per user. The user carries the hash pre-images, printed as 8-character values like `IZdB bqyH`. Logging in requires typing a "prefix password" as well as one randomly-queried hash-preimage.

OTPW rates poorly for usability: the prefix password means the scheme isn't *Memorywise-Effortless* or *Scalable-for-Users*; it also isn't *Nothing-to-Carry* because of the paper token. The typing of random passwords means the scheme also isn't *Physically-Effortless*, *Efficient-to-Use* or *Infrequent-Errors*. We do expect that the scheme is *Easy-to-Learn*, as typing in a numbered password upon request is only marginally more difficult than using text passwords. It is also *Easy-Recovery-from-Loss* as we expect most users can easily print a new sheet if needed.

Paper-based tokens are cheap and easy to deploy. We rate OTPW as non-*Accessible* because plain printing may be insufficient for visually-impaired users, though alternatives (e.g. braille) may be available. We consider the price of printing to be *Negligible-Cost-per-User*. While not *Server-Compatible*, the scheme is *Browser-Compatible*. Finally, OTPW has a mature open-source implementation, making it *Mature* and *Non-Proprietary*.

Though OTPW is designed to resist human observation compared to S/KEY, it isn't *Resilient-to-Physical-Observation* because the printed sheet of one-time codes can be completely captured by a camera. Otherwise, OTPW achieves all other security benefits. Because login codes are used only once and randomly generated, the scheme is *Resilient-to-Throttled-Guessing*, *Resilient-to-Unthrottled-Guessing* and *Resilient-to-Internal-Observation*.

It is *Resilient-to-Phishing* as it is impractical for a user to enter all of their secrets into a phishing website even if asked, and *Resilient-to-Theft* thanks to the prefix password. As a one-to-one scheme with different secrets for each server, it is *Resilient-to-Leaks-from-Other-Verifiers*, *No-Trusted-Third-Party* and *Unlinkable*. Finally, the typing required makes it *Requiring-Explicit-Consent*.

G. Hardware tokens: RSA SecurID

Hardware tokens store secrets in a dedicated tamper-resistant module carried by the user; the RSA SecurID [34] family of tokens is the long-established market leader. Here we refer to the simplest dedicated-hardware version, which has only a display and no buttons or I/O ports. Each instance of the device holds a secret "seed" known to the back-end. A cryptographically strong transform generates a new 6-digit code from this secret every 60 seconds. The current code is shown on the device's display. On enrollment, the user connects to the administrative back-end through a web interface, where he selects a PIN and where the pairing between username and token is confirmed. From then on, for authenticating, instead of username and password the user shall type username and "passcode" (concatenation of a static 4-digit PIN and the dynamic 6-digit code). RSA offers an SSO facility to grant access to several corporate resources with the same token; but we rate this scheme assuming there won't be a single SSO spanning all verifiers.

In March 2011 attackers compromised RSA's back-end database of seeds [35], which allowed them to predict the codes issued by any token. This reduced the security of each account to that of its PIN until the corresponding token was recalled and reissued.

The scheme is not *Memorywise-Effortless* nor *Scalable-for-Users* (it needs a new token and PIN per verifier). It's not *Physically-Effortless*, because the user must transcribe the passcode. It's simple enough to be *Easy-to-Learn*, but *Quasi-Efficient-to-Use* because of the transcription. We rate it as having *Quasi-Infrequent-Errors*, like passwords, though it might be slightly worse. It is not *Easy-Recovery-from-Loss*: the token must be revoked and a new one reissued.

The scheme is not *Accessible*: blind users cannot read the code off the token. No token-based scheme can offer *Negligible-Cost-per-User*. The scheme is not *Server-Compatible* (a new back-end is required) but it is *Browser-Compatible*. It is definitely *Mature*, but not *Non-Proprietary*.

As for security, because the code changes every minute, SecurID is *Resilient-to-Physical-Observation*, *Resilient-to-Targeted-Impersonation*, *Resilient-to-Throttled-Guessing* and *Resilient-to-Unthrottled-Guessing* (unless we also assume that the attacker broke into the server and stole the seeds). It is *Resilient-to-Internal-Observation*: we assume that dedicated devices can resist malware infiltration. It's *Resilient-to-Leaks-from-Other-Verifiers*, as different verifiers would have their own seeds; *Resilient-to-Phishing*, because captured passcodes expire after one minute; and *Resilient-to-Theft*, because the PIN is checked at the verifier, so guesses could be rate-limited. It's not *No-Trusted-Third-Party*, as demonstrated by the March 2011 attack, since RSA keeps the seed of each token. It's *Requiring-Explicit-Consent*, as the user must transcribe the passcode, and *Unlinkable* if each verifier requires its own token.

H. Mobile-Phone-based: Phoolproof

Phoolproof Phishing Prevention [36] is another token-based design, but one in which the token is a mobile phone with special code and crypto keys. It uses public key cryptography and an SSL-like authentication protocol and was designed to be as compatible as possible with existing systems.

Phoolproof was conceived as a system to secure banking transactions against phishing, not as a password replacement. The user selects a desired site from the whitelist on the phone; the phone talks wirelessly to the browser, causing the site to be visited; an end-to-end TLS-based mutual authentication ensues between the phone and the bank's site; the user must still type the banking website password into the browser. Thus the scheme is not *Memorywise-Effortless*, nor *Scalable-for-Users*. It has *Quasi-Nothing-to-Carry* (the mobile phone). It's not *Physically-Effortless* as one must type a password. We rate it *Easy-to-Learn*, perhaps generously, and *Quasi-Efficient-to-Use* as it requires both typing a password and fiddling with a phone. It's no better than passwords on *Quasi-Infrequent-Errors*, since it still uses one. The only recovery mechanism is revocation and reissue, so it doesn't have *Easy-Recovery-from-Loss*.

On deployability: it's *Quasi-Accessible* insofar as most disabled users, including blind people, can use a mobile phone too (note the user doesn't need to transcribe codes from the phone). We assume most users will already have a phone, though perhaps not one of the right type (with Java, Bluetooth etc), hence it has *Quasi-Negligible-Cost-per-User*. The scheme requires changes, albeit minor, to both ends, so it's *Quasi-Server-Compatible* but, by our definitions, not *Browser-Compatible* because it uses a browser plugin. It's not really *Mature* (only a research prototype), but it is *Non-Proprietary*.

On security: it's *Resilient-to-Physical-Observation*, *Resilient-to-Targeted-Impersonation*, *Resilient-to-Throttled-Guessing*, *Resilient-to-Unthrottled-Guessing* because, even after observing or guessing the correct password, the attacker can't authenticate unless he also steals the user's phone, which holds the cryptographic keys. It's *Quasi-Resilient-to-Internal-Observation* because malware must compromise both the phone (to capture the private keys) and the computer (to keylog the password). It's *Resilient-to-Leaks-from-Other-Verifiers* because the phone has a key pair per verifier, so credentials are not recycled. It's definitely *Resilient-to-Phishing*, the main design requirement of the

scheme. It's *Resilient-to-Theft* because possession of the phone is insufficient: the user still needs to type user ID and password in the browser (for additional protection against theft, the authors envisage an additional PIN or biometric to authenticate the user to the device; we are not rating this). The scheme is *No-Trusted-Third-Party* if we disregard the CA that certifies the TLS certificate of the bank. It's *Requiring-Explicit-Consent* because the user must type user ID and password. Finally it's *Unlinkable* because the phone has a different key pair for each verifier.

I. Biometrics: Fingerprint recognition

Biometrics [37] are the "what you are" means of authentication, leveraging the uniqueness of physical or behavioral characteristics across individuals. We discuss in detail *fingerprint* biometrics [38]; our summary table also rates *iris recognition* [39] and voiceprint biometrics [40]. In rating for our remote authentication application, and *biometric verification* ("Is this individual asserted to be Jane Doe really Jane Doe?"), we assume unsupervised biometric hardware as might be built into client devices, vs. verifier-provided hardware, e.g., at an airport supervised by officials.

Fingerprint biometrics offer usability advantages *Memorywise-Effortless*, *Scalable-for-Users*, *Easy-to-Learn*, and *Nothing-to-Carry* (no secrets need be carried; we charge elsewhere for client-side fingerprint readers not being currently universal). Current products are at best *Quasi-Physically-Effortless* and *Quasi-Efficient-to-Use* due to user experience of not *Infrequent-Errors* (the latter two worse than web passwords) and fail to offer *Easy-Recovery-from-Loss* (here equated with requiring an alternate scheme in case of compromise, or users becoming unable to provide the biometric for physical reasons).

Deployability is poor—we rate it at best *Quasi-Accessible* due to common failure-to-register biometric issues; not *Negligible-Cost-per-User* (fingerprint reader has a cost); neither *Server-Compatible* nor *Browser-Compatible*, needing both client and server changes; at best *Quasi-Mature* for unsupervised remote authentication; and not *Non-Proprietary*, typically involving proprietary hardware and/or software.

We rate the fingerprint biometric *Resilient-to-Physical-Observation* but serious concerns include easily fooling COTS devices, e.g., by lifting fingerprints from glass surfaces with gelatin-like substances [41], which we charge by rating not *Resilient-to-Targeted-Impersonation*. It is *Resilient-to-Throttled-Guessing*, but not *Resilient-to-Unthrottled-Guessing* for typical precisions used; estimated "effective equivalent key spaces" [9, page 2032] for fingerprint, iris and voice are 13.3 bits, 19.9 bits and 11.7 bits respectively. It is not *Resilient-to-Internal-Observation* (captured samples of static physical biometrics are subject to replay in unsupervised environments), not *Resilient-to-Leaks-from-Other-Verifiers*, not *Resilient-to-Phishing* (a serious concern as biometrics are by design supposed to be hard to change), and not *Resilient-to-Theft* (see above re: targeted impersonation). As a plus, it needs *No-Trusted-Third-Party* and is *Requiring-Explicit-Consent*. Physical biometrics are also a canonical example of schemes that are not *Unlinkable*.

V. Discussion

A clear result of our exercise is that no scheme we examined is perfect—or even comes close to perfect scores. The incumbent (traditional passwords) achieves all benefits on deployability, and one scheme (the CAP reader, discussed in the tech report [1]) achieves all in security, but no scheme achieves all usability benefits. Not a single scheme is dominant over passwords, i.e., does better on one or more benefits and does at least as well on all others. Almost all schemes do better than passwords in some criteria, but all are worse in others: as Table I shows, no row is free of red (horizontal) stripes.

Thus, the current state of the world is a Pareto equilibrium. Replacing passwords with any of the schemes examined is not a question of giving up an inferior technology for something unarguably better, but of giving up one set of compromises and trade-offs in exchange for another. For example, arguing that a hardware token like RSA SecurID is better than passwords implicitly assumes that the security criteria where it does better outweigh the usability and deployability criteria where it does worse. For accounts that require high assurance, security benefits may indeed outweigh the fact that the scheme doesn't offer *Nothing-to-Carry* nor *Negligible-Cost-per-User*, but this argument is less compelling for lower value accounts.

The usability benefits where passwords excel—namely, *Nothing-to-Carry*, *Efficient-to-Use*, *Easy-Recovery-from-Loss*—are where essentially all of the stronger security schemes need improvement. None of the paper token or hardware token schemes achieves even two of these three. In expressing frustration with the continuing dominance of passwords, many security experts presumably view these two classes of schemes to be sufficiently usable to justify a switch from passwords. The web sites that crave user traffic apparently disagree.

Some sets of benefits appear almost incompatible, e.g., the pair (*Memorywise-Effortless*, *Nothing-to-Carry*) is achieved only by biometric schemes. No schemes studied achieve (*Memorywise-Effortless*, *Resilient-to-Theft*) fully, nor (*Server-Compatible*, *Resilient-to-Internal-Observation*) or (*Server-Compatible*, *Resilient-to-Leaks-from-Other-Verifiers*), though several almost do. Note that since compatibility with existing servers almost assures a static replayable secret, to avoid its security implications, many proposals abandon being *Server-Compatible*.

A. Rating categories of schemes

Password managers offer advantages over legacy passwords in selected usability and security aspects without

Category	Scheme	Described in section	Reference	Usability							Deployability						Security											
				Memorywise-Effortless	*Scalable-for-Users*	*Nothing-to-Carry*	*Physically-Effortless*	*Easy-to-Learn*	*Efficient-to-Use*	*Infrequent-Errors*	*Easy-Recovery-from-Loss*	*Accessible*	*Negligible-Cost-per-User*	*Server-Compatible*	*Browser-Compatible*	*Mature*	*Non-Proprietary*	*Resilient-to-Physical-Observation*	*Resilient-to-Targeted-Impersonation*	*Resilient-to-Throttled-Guessing*	*Resilient-to-Unthrottled-Guessing*	*Resilient-to-Internal-Observation*	*Resilient-to-Leaks-from-Other-Verifiers*	*Resilient-to-Phishing*	*Resilient-to-Theft*	*No-Trusted-Third-Party*	*Requiring-Explicit-Consent*	*Unlinkable*
(Incumbent)	Web passwords	III	[13]			●		●	●	○	●	●	●	●	●	●	●		○						●	●	●	●
Password managers	Firefox	IV-A	[22]	○	●	○	○	●	●	●		●	●	●		●	●	○	○					●	●	●	●	●
	LastPass		[42]	○	●	○	○	●	●	●	○	●	○	●		●		○	○	○		○	●	●	●	●	●	●
Proxy	URRSA	IV-B	[5]	●				●		○			●	○	●				○			○		●			●	●
	Impostor		[23]	○	●	●		●			●	●	●	●	○		●	●	○			○		●	●			●
Federated	OpenID	IV-C	[27]	○	●	●	○	○	●	●	●	●	●		●	●	●	○	○	○	○		●		●		●	
	Microsoft Passport		[43]	○	●	●	○	●	●	●	●	●			●	●		○	○	○	○		●		●		●	
	Facebook Connect		[44]	○	●	●	○	●	●	●	●	●	●		●	●		○	○	○	○		●		●			
	BrowserID		[45]	○	●	●	○	●	●	●	●	●	●		○	○	●	○	○	○	○		●		●		●	
	OTP over email		[46]	○	●	●		●		●	●	●	●		●		●	○	○	○	○		●	●	●		●	
Graphical	PCCP	IV-D	[7]			●		●	○	○	●		●		●		●		●	○			●	●	●	●	●	●
	PassGo		[47]			●		●	○	○	●		●		●	○	●		●						●	●	●	●
Cognitive	GrIDsure (original)	IV-E	[30]			●		●	○	○	●		●		●				●						●	●	●	●
	Weinshall		[48]			●							●		●		●	○	●				●	●	●	●	●	●
	Hopper Blum		[49]			●						●	●		●		●	○	●				●	●	●	●	●	●
	Word Association		[50]			●		●	●	○	○	●	●		●		●								●	●	●	●
Paper tokens	OTPW	IV-F	[33]					●			●		●		●	●	●		●	●	●	●	●	●	●	●	●	●
	S/KEY		[32]	●				●		○	●		●		●	●	●		●	●	●	●	●	○	●	●	●	●
	PIN+TAN		[51]					●		○	○		○		●	●	●		●	●	●	●	●	●	○	●	●	●
Visual crypto	PassWindow		[52]	●									○		●	●	○	●	●	●	●	○	●	●		●	●	●
Hardware tokens	RSA SecurID	IV-G	[34]					●	○	○					●	●	●	●	●	●	●	●	●	●	●		●	●
	YubiKey		[53]					●	○	○		●			●	●	●	●	●	●	●	●	●	●			●	●
	IronKey		[54]	○	●		○	○	○	○		●		●	●	●	●	○			○		●	●	●	●	●	●
	CAP reader		[55]					●	○	○					●	●	●	●	●	●	●	●	●	●	●	●	●	●
	Pico		[8]	●	●		●		○	○						●	●	●	●	●	●	●	●	○	●	●	●	●
Phone-based	Phoolproof	IV-H	[36]			○		●	○	○		○	○	○			●	●	●	●	○	●	●	●	●	●	●	●
	Cronto		[56]			○		●	○	○			○		●	●	●	●	●	●	○	●	●	●	●	●	●	●
	MP-Auth		[6]			○		●	○		○	○	○			●		○					●	●	●	●	●
	OTP over SMS			●	●	○		●		○	○	○			●	●	●	●	●	●	○	●	●	○	●	●	●	●
	Google 2-Step		[57]			○		●	○	○	○	○			●	●		○	○	●		●	●	●	●	●	●	●
Biometric	Fingerprint	IV-I	[38]	●	●	●	○	●	○			○			○		●	●		●						●	●	
	Iris		[39]	●	●	●	○	●	○			○			○		●	●		●						●	○	
	Voice		[40]	●	●	●	○	●	○			○	○		○	○	●			○						●	●	
Recovery	Personal knowledge		[58]	○		●		●	●	○	●	●	●		●	●	●								●	●	●	●
	Preference-based		[59]	○		●		●	○	●	○	●	●		●			○						●	●	●	●	●
	Social re-auth.		[60]			●		●		●		●	●		●	○	○		●	●	○	○	●	●	●		●	○

●= offers the benefit; ○= almost offers the benefit; *no circle* = does not offer the benefit.

▒= better than passwords; ▒= worse than passwords; *no background pattern* = no change.

We group related schemes into categories. For space reasons, in the present paper we describe at most one representative scheme per category; the companion technical report [1] discusses all schemes listed.

Table I

COMPARATIVE EVALUATION OF THE VARIOUS SCHEMES WE EXAMINED

losing much. They could become a staple of users' coping strategies if passwords remain widespread, enabling as a major advantage the management of an ever-increasing number of accounts (*Scalable-for-Users*). However, the underlying technology remains replayable, static (mainly user-chosen) passwords.

Federated schemes are particularly hard to grade. Proponents note that security is good if authentication to the identity provider (IP) is done with a strong scheme (e.g., one-time passwords or tokens). However in this case usability is inherited from that scheme and is generally poor, per Table I. This also reduces federated schemes to be a placeholder for a solution rather than a solution itself. If authentication to the IP relies on passwords, then the resulting security is only a little better than that of passwords themselves (with fewer password entry instances exposed to attack).

Graphical passwords can approach text passwords on usability criteria, offering some security gain, but static secrets are replayable and not *Resilient-to-Internal-Observation*. Despite adoption for device access-control on some touch-screen mobile devices, for remote web authentication the advantages appear insufficient to generally displace a firmly-entrenched incumbent.

Cognitive schemes show slender improvement on the security of passwords, in return for worse usability. While several schemes attempt to achieve *Resilient-to-Internal-Observation*, to date none succeed: the secret may withstand one observation or two [61], but seldom more than a handful [62]. The apparently inherent limitations [63], [64] of cognitive schemes to date lead one to question if the category can rise above one of purely academic interest.

The hardware token, paper token and phone-based categories of schemes fare very well in security, e.g., most in Table I are *Resilient-to-Internal-Observation*, easily beating other classes. However, that S/KEY and SecurID have been around for decades and have failed to slow down the inexorable rise of passwords suggests that their drawbacks in usability (e.g., not *Scalable-for-Users*, nor *Nothing-to-Carry*, nor *Efficient-to-Use*) and deployability (e.g., hardware tokens are not *Negligible-Cost-per-User*) should not be over-looked. Less usable schemes can always be mandated, but this is more common in situations where a site has a de facto monopoly (e.g., employee accounts or government sites) than where user acceptance matters. Experience shows that the large web-sites that compete for both traffic and users are reluctant to risk bad usability [16]. Schemes that are less usable than passwords face an uphill battle in such environments.

Biometric schemes have mixed scores on our usability metrics, and do poorly in deployability and security. As a major issue, physical biometrics being inherently non-*Resilient-to-Internal-Observation* is seriously compounded by biometrics missing *Easy-Recovery-from-Loss* as well, with re-issuance impossible [9]. Thus, e.g., if malware captures the digital representation of a user's iris, possible replay makes the biometric no longer suitable in unsupervised environments. Hence despite security features appropriate to control access to physical locations under the supervision of suitable personnel, biometrics aren't well suited for unsupervised web authentication where client devices lack a trusted input path and means to verify that samples are live.

B. Extending the benefits list

Our list of benefits is not complete, and indeed, any such list could always be expanded. We did not include resistance to active-man-in-the-middle, which a few examined schemes may provide, or to relay attacks, which probably none of them do. However, tracking all security goals, whether met or not, is important and considering benefits that indicate resistance to these (and additional) attacks is worthwhile.

Continuous authentication (with ongoing assurances rather than just at session start, thereby addressing session hijacking) is a benefit worth considering, although a goal of few current schemes. Positive user affectation (how pleasant users perceive use of a scheme to be) is a standard usability metric we omitted; unfortunately, the literature currently lacks this information for most schemes. The burden on the end-user in migrating from passwords (distinct from the deployability costs of modifying browser and server infrastructure) is another important cost—both the one-time initial setup and per-account transition costs. While ease of resetting and revoking credentials falls within *Easy-Recovery-from-Loss*, the benefit does not include user and system aspects related to ease of renewing credentials that expire within normal operations (excluding loss). Other missing cost-related benefits are low cost for initial setup (including infrastructure changes by all stakeholders); low cost for ongoing administration, support and maintenance; and low overall complexity (how many inter-related "moving parts" a system has). We don't capture continued availability under denial-of-service attack, ease of use on mobile devices, nor the broad category of economic and business effects—e.g., the lack of incentive to be a relying party is cited as a main reason for OpenID's lack of adoption [28].

We have not attempted to capture these and other benefits in the present paper, though all fit into the framework and could be chosen by others using this methodology. Alas, many of these raise a difficulty: assigning ratings might be even more subjective than for existing benefits.

C. Additional nuanced ratings

We considered, but did not use, a "fatal" rating to indicate that a scheme's performance on a benefit is so poor that the scheme should be eliminated from serious consideration. For example, the 2–3 minutes required for authentication using the Weinshall or Hopper-Blum schemes may make them "fatally-non-*Efficient-to-Use*", likely preventing widespread adoption even if virtually all other benefits were provided.

We decided against this because for many properties, it isn't clear what level of failure to declare as fatal.

We also considered a "power" rating to indicate that a scheme optionally enables a benefit for power users—e.g., OpenID could be rated "amenable-to-*No-Trusted-Third-Party*" as users can run their own identity servers, in contrast to Facebook Connect or Microsoft Passport. The popularity of webmail-based password reset indicates most users accede to a heavily-trusted third party for their online identities already, so "amenable-to" may suffice for adoption. OpenID is arguably amenable to every security benefit for power users, but doesn't provide them for common users who use text passwords to authenticate to their identity provider. However, as one could argue for an amenable-to rating for many properties of many schemes, we maintained focus on properties provided by default to all users.

D. Weights and finer-grained scoring

We reiterate a caution sounded at the end of Section II: the benefits chosen as metrics are not all of equal weight. The importance of any particular benefit depends on target use and threat environment. While one could assign weights to each column to compute numerical scores for each scheme, providing exact weights is problematic and no fixed values would suit all scenarios; nonetheless, our framework allows such an endeavour. For finer-grained evaluation, table cell scores like *partially* could also be allowed beyond our very coarse {*no*, *almost*, *yes*} quantization, to further delineate similar schemes. This has merit but brings the danger of being "precisely wrong", and too fine a granularity adds to the difficulty of scoring schemes consistently. There will be the temptation to be unrealistically precise ("If scheme X gets 0.9 for this benefit, then scheme Y should get at most 0.6"), but this demands the ability to maintain a constant level of precision *repeatably* across all cells.

We have resisted the temptation to produce an aggregate score for each scheme (e.g., by counting the number of benefits achieved), or to rank the schemes. As discussed above, fatal failure of a single benefit or combined failure of a pair of benefits (e.g., not being *Resilient-to-Internal-Observation* and fatally failing *Easy-Recovery-from-Loss* for biometrics) may eliminate a scheme from consideration. Thus, seeking schemes purely based on high numbers of benefits could well prove but a distraction.

Beyond divergences of judgement, there will no doubt be errors in judgement in scoring. The table scoring methodology must include redundancy and cross-checks sufficient to catch most such errors. (Our exercise involved one author initially scoring a scheme row, co-authors verifying the scores, and independently, cross-checks within columns to calibrate individual benefit ratings across schemes; useful clarifications of benefit definitions often resulted.) Another danger in being "too precise" arises from scoring on second-hand data inferred from papers. Coarsely-quantized but self-consistent scores are likely better than inconsistent ones.

On one hand, it could be argued that different application domains (e.g., banking vs. gaming) have different requirements and that therefore they ought to assign different weights to the benefits, resulting in a different choice of optimal scheme for each domain. However on the other hand, to users, a proliferation of schemes is in itself a failure: the meta-scheme of "use the best scheme for each application" will score rather poorly on *Scalable-for-Users*, *Easy-to-Learn* and perhaps a few other usability benefits.

E. Combining schemes

Pairs of schemes that complement each other well in a two-factor arrangement might be those where *both* achieve good scores in usability and deployability and *at least one* does so in security—so a combined scheme might be viewed as having the AND of the usability-deployability scores (i.e., the combination does not have a particular usability or deployability benefit unless both of the schemes do) and the OR of the security scores (i.e., the combination has the security benefit if either of the schemes do). An exception would appear to be the usability benefit *Scalable-for-Users* which a combination might inherit from either component.

However, this is necessarily just a starting point for the analysis: it is optimistic to assume that two-component schemes always inherit benefits in this way. Wimberly and Liebrock [65] observed that the presence of a second factor caused users to pick much weaker passwords than if passwords alone were used to protect an account—as predicted by Adams's "risk thermostat" model [66]. Thus, especially where user choice is involved, there can be an erosion of the efficacy of one protection when a second factor is known to be in place. Equally, defeating one security mechanism may also make it materially easier to defeat another. We rated, e.g., Phoolproof *Quasi-Resilient-to-Internal-Observation* because it requires an attacker to compromise both a PC and a mobile device. However, malware has already been observed in the wild which leverages a compromised PC to download further malware onto mobile devices plugged into the PC for a software update [67].

See O'Gorman [9] for suggested two-factor combinations of biometrics, passwords, and tokens, for various applications (e.g., combining a hardware token with a biometric). Another common suggestion is pairing a federated scheme with a higher-security scheme, e.g., a hardware token.

VI. Concluding Remarks

The concise overview offered by Table I allows us to see high level patterns that might otherwise be missed. We could at this stage draw a variety of conclusions and note, for example, that graphical and cognitive schemes offer only minor improvements over passwords and thus have little hope of displacing them. Or we could note that most of

the schemes with substantial improvements in both usability and security can be seen as incarnations of Single-Sign-On (including in this broad definition not only federated schemes but also "local SSO" systems [26] such as password managers or Pico). Having said that, we expect the long-term scientific value of our contribution will lie not as much in the raw data distilled herein, as in the methodology by which it was assembled. A carefully crafted benefits list and coherent methodology for scoring table entries, despite inevitable (albeit instructive) disagreements over fine points of specific scores, allows principled discussions about high level conclusions.

That a Table I scheme (the CAP reader) scored full marks in security does not at all suggest that its real-world security is perfect—indeed, major issues have been found [55]. This is a loud warning that it would be unwise to read absolute verdicts into these scores. Our ratings are useful and we stand by them, but they are not a substitute for independent critical analysis or for considering aspects we didn't rate, such as vulnerability to active man-in-the-middle attacks.

We note that the ratings implied by scheme authors in original publications are often not only optimistic, but also incomplete. Proponents, perhaps subconsciously, often have a biased and narrow view of what benefits are relevant. Our framework allows a more objective assessment.

In closing we observe that, looking at the green (vertical) and red (horizontal) patterns in Table I, most schemes do better than passwords on security—as expected, given that inventors of alternatives to passwords tend to come from the security community. Some schemes do better and some worse on usability—suggesting that the community needs to work harder there. But *every* scheme does worse than passwords on deployability. This was to be expected given that the first four deployability benefits are defined with explicit reference to what passwords achieve and the remaining two are natural benefits of a long-term incumbent, but this uneven playing field reflects the reality of a decentralized system like the Internet. Marginal gains are often not sufficient to reach the activation energy necessary to overcome significant transition costs, which may provide the best explanation of why we are likely to live considerably longer before seeing the funeral procession for passwords arrive at the cemetery.

Acknowledgments

The authors thank the anonymous reviewers whose comments helped improve the paper greatly. Joseph Bonneau is supported by the Gates Cambridge Trust. Paul C. van Oorschot is Canada Research Chair in Authentication and Computer Security, and acknowledges NSERC for funding the chair and a Discovery Grant; partial funding from NSERC ISSNet is also acknowledged. This work grew out of the Related Work section of Pico [8].

References

[1] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," University of Cambridge Computer Laboratory, Tech Report 817, 2012, www.cl.cam.ac.uk/techreports/UCAM-CL-TR-817.html.

[2] R. Morris and K. Thompson, "Password security: a case history," *Commun. ACM*, vol. 22, no. 11, pp. 594–597, 1979.

[3] A. Adams and M. Sasse, "Users Are Not The Enemy," *Commun. ACM*, vol. 42, no. 12, pp. 41–46, 1999.

[4] C. Herley and P. C. van Oorschot, "A research agenda acknowledging the persistence of passwords," *IEEE Security & Privacy*, vol. 10, no. 1, pp. 28–36, 2012.

[5] D. Florêncio and C. Herley, "One-Time Password Access to Any Server Without Changing the Server," *ISC 2008, Taipei*.

[6] M. Mannan and P. C. van Oorschot, "Leveraging personal devices for stronger password authentication from untrusted computers," *Journal of Computer Security*, vol. 19, no. 4, pp. 703–750, 2011.

[7] S. Chiasson, E. Stobert, A. Forget, R. Biddle, and P. C. van Oorschot, "Persuasive cued click-points: Design, implementation, and evaluation of a knowledge-based authentication mechanism," *IEEE Trans. on Dependable and Secure Computing*, vol. 9, no. 2, pp. 222–235, 2012.

[8] F. Stajano, "Pico: No more passwords!" in *Proc. Sec. Protocols Workshop 2011*, ser. LNCS, vol. 7114. Springer.

[9] L. O'Gorman, "Comparing passwords, tokens, and biometrics for user authentication," *Proceedings of the IEEE*, vol. 91, no. 12, pp. 2019–2040, December 2003.

[10] K. Renaud, "Quantification of authentication mechanisms: a usability perspective," *J. Web Eng.*, vol. 3, no. 2, pp. 95–123, 2004.

[11] R. Biddle, S. Chiasson, and P. C. van Oorschot, "Graphical Passwords: Learning from the First Twelve Years," *ACM Computing Surveys*, vol. 44, no. 4, 2012.

[12] J. Nielsen and R. Mack, *Usability Inspection Methods*. John Wiley & Sons, Inc, 1994.

[13] J. Bonneau and S. Preibusch, "The password thicket: technical and market failures in human authentication on the web," in *Proc. WEIS 2010*, 2010.

[14] J. Bonneau, "The science of guessing: analyzing an anonymized corpus of 70 million passwords," *IEEE Symp. Security and Privacy*, May 2012.

[15] K. Fu, E. Sit, K. Smith, and N. Feamster, "Dos and don'ts of client authentication on the web," in *Proc. USENIX Security Symposium*, 2001.

[16] D. Florêncio and C. Herley, "Where Do Security Policies Come From?" in *ACM SOUPS 2010: Proc. 6th Symp. on Usable Privacy and Security*.

[17] L. Falk, A. Prakash, and K. Borders, "Analyzing websites for user-visible security design flaws," in *ACM SOUPS 2008*, pp. 117–126.

[18] S. Gaw and E. W. Felten, "Password Management Strategies for Online Accounts," in *ACM SOUPS 2006: Proc. 2nd Symp. on Usable Privacy and Security*, pp. 44–55.

[19] D. Florêncio and C. Herley, "A large-scale study of web password habits," in *WWW '07: Proc. 16th International Conf. on the World Wide Web*. ACM, 2007, pp. 657–666.

[20] D. Balzarotti, M. Cova, and G. Vigna, "ClearShot: Eavesdropping on Keyboard Input from Video," in *IEEE Symp. Security and Privacy*, 2008, pp. 170–183.

[21] B. Kaliski, *RFC 2898: PKCS #5: Password-Based Cryptography Specification Version 2.0*, IETF, September 2000.

[22] Mozilla Firefox, ver. 10.0.2, www.mozilla.org/.

[23] A. Pashalidis and C. J. Mitchell, "Impostor: A single sign-on system for use from untrusted devices," *Proc. IEEE Globecom*, 2004.

[24] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Commun. ACM*, vol. 21, pp. 993–999, December 1978.

[25] J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," United States, 1993.

[26] A. Pashalidis and C. J. Mitchell, "A Taxonomy of Single Sign-On Systems," in *Proc. ACISP 2003, Information Security and Privacy, 8th Australasian Conference*. Springer LNCS 2727, 2003, pp. 249–264.

[27] D. Recordon and D. Reed, "OpenID 2.0: a platform for user-centric identity management," in *DIM '06: Proc. 2nd ACM Workshop on Digital Identity Management*, 2006, pp. 11–16.

[28] S.-T. Sun, Y. Boshmaf, K. Hawkey, and K. Beznosov, "A billion keys, but few locks: the crisis of web single sign-on," *Proc. NSPW 2010*, pp. 61–72.

[29] B. Laurie, "OpenID: Phishing Heaven," January 2007, www.links.org/?p=187.

[30] R. Jhawar, P. Inglesant, N. Courtois, and M. A. Sasse, "Make mine a quadruple: Strengthening the security of graphical one-time pin authentication," in *Proc. NSS 2011*, pp. 81–88.

[31] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, pp. 770–772, 1981.

[32] N. Haller and C. Metz, "RFC 1938: A One-Time Password System," 1998.

[33] M. Kuhn, "OTPW — a one-time password login package," 1998, www.cl.cam.ac.uk/~mgk25/otpw.html.

[34] RSA, "RSA SecurID Two-factor Authentication," 2011, www.rsa.com/products/securid/sb/10695_SIDTFA_SB_0210.pdf.

[35] P. Bright, "RSA finally comes clean: SecurID is compromised," Jun. 2011, arstechnica.com/security/news/2011/06/rsa-finally-comes-clean-securid-is-compromised.ars.

[36] B. Parno, C. Kuo, and A. Perrig, "Phoolproof Phishing Prevention," in *Proc. Fin. Crypt. 2006*, pp. 1–19.

[37] A. K. Jain, A. Ross, and S. Pankanti, "Biometrics: a tool for information security," *IEEE Transactions on Information Forensics and Security*, vol. 1, no. 2, pp. 125–143, 2006.

[38] A. Ross, J. Shah, and A. K. Jain, "From Template to Image: Reconstructing Fingerprints from Minutiae Points," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 4, pp. 544–560, 2007.

[39] J. Daugman, "How iris recognition works," *IEEE Trans. Circuits Syst. Video Techn.*, vol. 14, no. 1, pp. 21–30, 2004.

[40] P. S. Aleksic and A. K. Katsaggelos, "Audio-Visual Biometrics," *Proc. of the IEEE*, vol. 94, no. 11, pp. 2025–2044, 2006.

[41] T. Matsumoto, H. Matsumoto, K. Yamada, and S. Hoshino, "Impact of artificial "gummy" fingers on fingerprint systems," in *SPIE Conf. Series*, vol. 4677, Apr. 2002, pp. 275–289.

[42] LastPass, www.lastpass.com/.

[43] D. P. Kormann and A. D. Rubin, "Risks of the Passport single signon protocol," *Computer Networks*, vol. 33, no. 1–6, 2000.

[44] "Facebook Connect," 2011, www.facebook.com/advertising/?connect.

[45] M. Hanson, D. Mills, and B. Adida, "Federated Browser-Based Identity using Email Addresses," *W3C Workshop on Identity in the Browser*, May 2011.

[46] T. W. van der Horst and K. E. Seamons, "Simple Authentication for the Web," in *Intl. Conf. on Security and Privacy in Communications Networks*, 2007, pp. 473–482.

[47] H. Tao, "Pass-Go, a New Graphical Password Scheme," Master's thesis, School of Information Technology and Engineering, University of Ottawa, June 2006.

[48] D. Weinshall, "Cognitive Authentication Schemes Safe Against Spyware (Short Paper)," in *IEEE Symposium on Security and Privacy*, May 2006.

[49] N. Hopper and M. Blum, "Secure human identification protocols," *ASIACRYPT 2001*, pp. 52–66, 2001.

[50] S. Smith, "Authenticating users by word association," *Computers & Security*, vol. 6, no. 6, pp. 464–470, 1987.

[51] A. Wiesmaier, M. Fischer, E. G. Karatsiolis, and M. Lippert, "Outflanking and securely using the PIN/TAN-System," *CoRR*, vol. cs.CR/0410025, 2004.

[52] "PassWindow," 2011, www.passwindow.com.

[53] Yubico, "The YubiKey Manual, v. 2.0," 2009, static.yubico.com/var/uploads/YubiKey_manual-2.0.pdf.

[54] Ironkey, www.ironkey.com/internet-authentication.

[55] S. Drimer, S. J. Murdoch, and R. Anderson, "Optimised to Fail: Card Readers for Online Banking," in *Financial Cryptography and Data Security*, 2009, pp. 184–200.

[56] Cronto, www.cronto.com/.

[57] Google Inc., "2-step verification: how it works," 2012, www.google.com/accounts.

[58] S. Schechter, A. J. B. Brush, and S. Egelman, "It's no secret: Measuring the security and reliability of authentication via 'secret' questions," in *IEEE Symp. Security and Privacy*, 2009, pp. 375–390.

[59] M. Jakobsson, L. Yang, and S. Wetzel, "Quantifying the Security of Preference-based Authentication," in *ACM DIM 2008: 4th Workshop on Digital Identity Management*.

[60] J. Brainard, A. Juels, R. L. Rivest, M. Szydlo, and M. Yung, "Fourth-factor authentication: somebody you know," in *ACM CCS 2006*, pp. 168–178.

[61] D. Weinshall, "Cognitive Authentication Schemes Safe Against Spyware," *IEEE Symp. Security and Privacy*, 2006.

[62] P. Golle and D. Wagner, "Cryptanalysis of a Cognitive Authentication Scheme," *IEEE Symp. Security and Privacy*, 2007.

[63] B. Coskun and C. Herley, "Can "Something You Know" be Saved?" *ISC 2008, Taipei.*

[64] Q. Yan, J. Han, Y. Li, and H. Deng, "On limitations of designing usable leakage-resilient password systems: Attacks, principles and usability." Proc. NDSS, 2012.

[65] H. Wimberly and L. M. Liebrock, "Using Fingerprint Authentication to Reduce System Security: An Empirical Study," in *IEEE Symp. Security and Privacy*, 2011, pp. 32–46.

[66] J. Adams, "Risk and morality: three framing devices," in *Risk and Morality*, R. Ericson and A. Doyle, Eds. University of Toronto Press, 2003.

[67] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *ACM SPSM 2011: 1st Workshop on Security and Privacy in Smartphones and Mobile Devices*, pp. 3–14.

UI Class
Reading

12/2

- learnability
- efficency
- safety
  - errors recoverable

- Fitts Law
- Tunneling
- Tracing the values
- # of items can see at once

Can ya learn by watching
video of how to
put on wax

mental models
Consistency

(2)

metaphores

affordance

feedback

training in JS

have ya try it

Chunking

7±2 chunks

~10 sec

p204

Fitt's Law

Pointing + steering

tunnel

p215

(3)

Speed - accuracy tradeoff 297

Heuristic eval reports ~360

Draft 12/2 MP

Novel Password Systems Where Enter Derivation of Password instead of Actual Password

6.858 Final Project

Michael Plasmeier <theplaz>
Jonathan Wang <jwang7>
Miguel Flores <mflores>

Motivation

The problem with many password systems is that users must type their entire, full password each time they log on. This makes the password vulnerable to key logging and interception during transmission.

We explore systems in which the user does not enter their direct password, but a derivation of the password **which changes on each log in**. The user proves that he or she knows the password without subsequently ever providing the password itself.

ING Password Keyboard

A simple example is ING Direct's PIN pad. Under ING's system, the user enters the letters corresponding to their PIN instead of the PIN itself. The mapping between numbers and letters is randomly generated on every log in. This method does not survive an attack where the attacker has access to the mapping, but it does prevent simple keylogging.

Figure 1 ING's Pin Pad. The user enters the letters corresponding to their PIN in the box.

Description of System (Plaz)

Original Off the Grid

We were inspired by the "Off the Grid" system from the Gibson Research Corporation.[1] The "Off the Grid" proposal is designed to allow users to use a personal printed paper grid to encipher the domain name of the website they are currently on into a string of psudeo-random characters.

The Off the Grid system works entirely on the user's side. Websites do not need to do anything to support Off the Grid.

To use Off the Grid, the user first generates a grid from a grid-providing website such as https://www.grc.com/offthegrid.htm. This website generates a grid using client-side scripting (ie. JavaScript) to generate the grid on the user's machine. The user then prints the grid onto a sheet of letter paper. At this point the Grid is offline and thus impossible to access by malware. As an alternative, there is at least one application for Android which produces and stores a grid; however, the grid is now accessible to malware on the Android phone which is able to defeat the inter-process sandboxing.

The grid that is generated is a Latin Square. A Latin Square is an $n \times n$ array filled with n different symbols, each occurring exactly once in each row and exactly once in each column.[2] The most famous Latin Square is the popular puzzle game Sudoku. (Note however, that we do not divide up the grid into 9 smaller 3x3 mini-squares in which each symbol must be unique). For example, here is a 11x11 Latin Square with 11 alphabetic characters:

g	e	a	m	o	n	z	k	i	r	c
k	a	n	c	m	z	o	r	g	i	e
n	k	c	z	a	m	r	g	o	e	i
z	o	i	a	n	g	e	c	r	k	m
m	r	z	n	g	a	k	i	e	c	o
a	g	e	i	z	r	n	o	c	m	k
r	c	g	k	e	i	m	n	z	o	a
e	n	k	g	i	o	c	z	m	a	r
i	m	o	e	r	c	g	a	k	n	k
c	i	r	o	k	e	a	m	n	z	g
o	z	m	r	c	k	i	e	a	g	n

Figure 2

Once the user has a grid, they use the grid to create or change the password for each website. The Off the Grid specification has a number of variants, but we will use the base variant described on the GRC website.

In the Off the Grid specification, the user traces the name of the website twice to provide additional entropy. In the start of the first phase, the user always starts along the first row of the grid.

---

[1] https://www.grc.com/offthegrid.htm and associated pages. Is still marked as "Work in Progress;" Retrieved 12/2/2012

[2] http://en.wikipedia.org/wiki/Latin_square

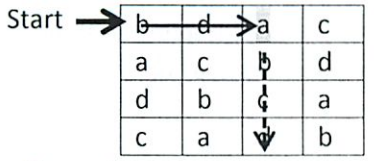


Figure 3

The user then traces out the first 6 characters of the domain name. 6 characters was chosen by the author to provide a 12 character password, which the author chose to balance ease of use with entropy. Again, a user may choose their own scheme. The user alternates between looking horizontally and vertically.

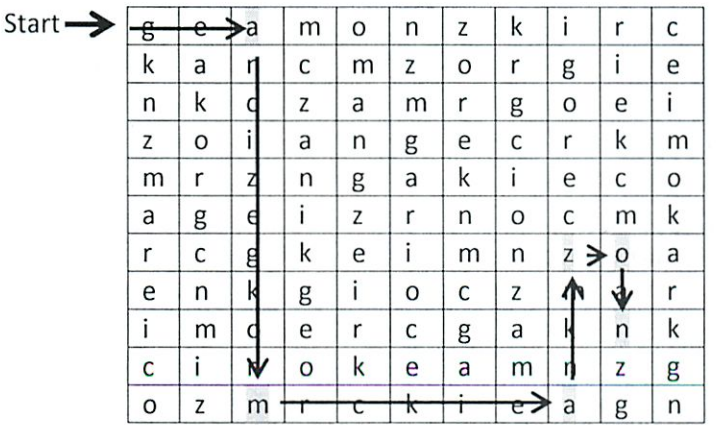


Figure 4

In the second phase, the user starts at the character that they ended with at the end of Phase 1. The user then two more characters from the grid in the same direction of travel. The user then appends those two characters to their password.

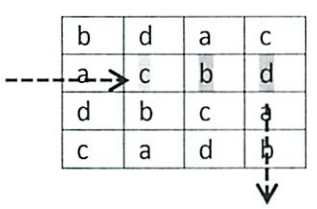


Figure 5 The user arrives at c traveling to the right. The user appends the next two characters "bd" to their password, and then continues up/down from the last character they read "d".

The user wraps around if their characters go off the grid.

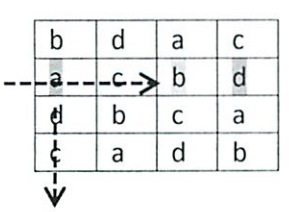
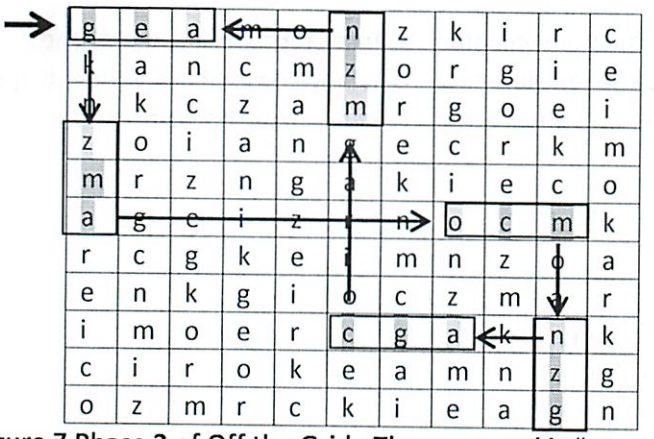


Figure 6 The user arrives at b traveling to the right. The user appends the next two characters to their password, wrapping around if they go off the edge of the grid. Here those characters are "da". The user then continues up/down from the last character "a".

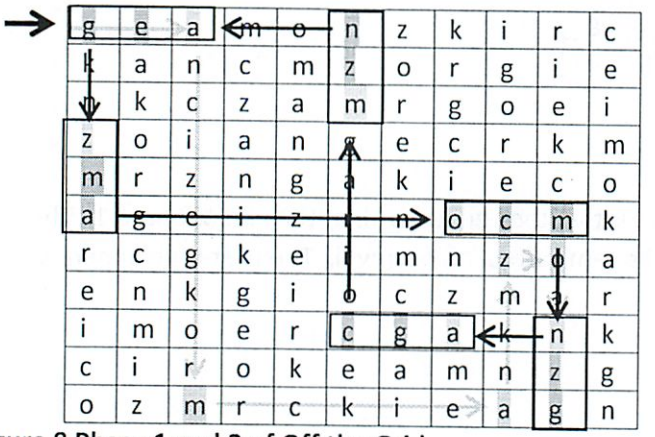
For example here is Phase 2 of our Amazon example.



g	e	a	m	o	n	z	k	i	r	c
l	a	n	c	m	z	o	r	g	i	e
v	k	c	z	a	m	r	g	o	e	i
z	o	i	a	n	g	e	c	r	k	m
m	r	z	n	g	a	k	i	e	c	o
a	g	e	i	z	l	n	o	c	m	k
r	c	g	k	e	i	m	n	z	d	a
e	n	k	g	i	o	c	z	m	a	r
i	m	o	e	r	c	g	a	k	n	k
c	i	r	o	k	e	a	m	n	z	g
o	z	m	r	c	k	i	e	a	g	n

Figure 7 Phase 2 of Off the Grid. The password is "gaznegmacmzg"

Here are Phase 1 and Phase 2.



g	e	a	m	o	n	z	k	i	r	c
l	a	n	c	m	z	o	r	g	i	e
v	k	c	z	a	m	r	g	o	e	i
z	o	i	a	n	g	e	c	r	k	m
m	r	z	n	g	a	k	i	e	c	o
a	g	e	i	z	l	n	o	c	m	k
r	c	g	k	e	i	m	n	z	d	a
e	n	k	g	i	o	c	z	m	a	r
i	m	o	e	r	c	g	a	k	n	k
c	i	r	o	k	e	a	m	n	z	g
o	z	m	r	c	k	i	e	a	g	n

Figure 8 Phase 1 and 2 of Off the Grid.

To log in, the user retraces exactly the same steps as when creating a password. This means the password is exactly the same for each domain. This is an obvious requirement for a system designed to fit within the existing password infrastructure. However, we wanted to explore ideas in which the user does not enter the same password each time.

Tracer: Trace the Grid

Name of system: Tracer or Cross Password?

We wanted to design a system similar to the Off the Grid system, but where the password the user transmits over the network is different each time. With this system, the website presents the user with a grid and the user enters only a deviation of their password.

When the user creates an account, he provides his or her password to the webserver. The user may use characters from the lower case Latin alphabet [a...z]. The password may not have consecutive

repeating characters, for example, "aardvark". The password is stored on the server such that the plain text can be accessed in order to verify the trace.

When the user logs in, the server randomly generates a 26x26 Latin square with the characters [a...z] called the *Grid*. The server also randomly selects a start row or column called the *start location*. The server transmits this Grid to the user. The Grid and the start location are unique for each log in. The server stores the Grid and start location in temporary state and provides a pointer to this state called the *token* to the user. The user's browser returns the token to the server on each log in attempt.

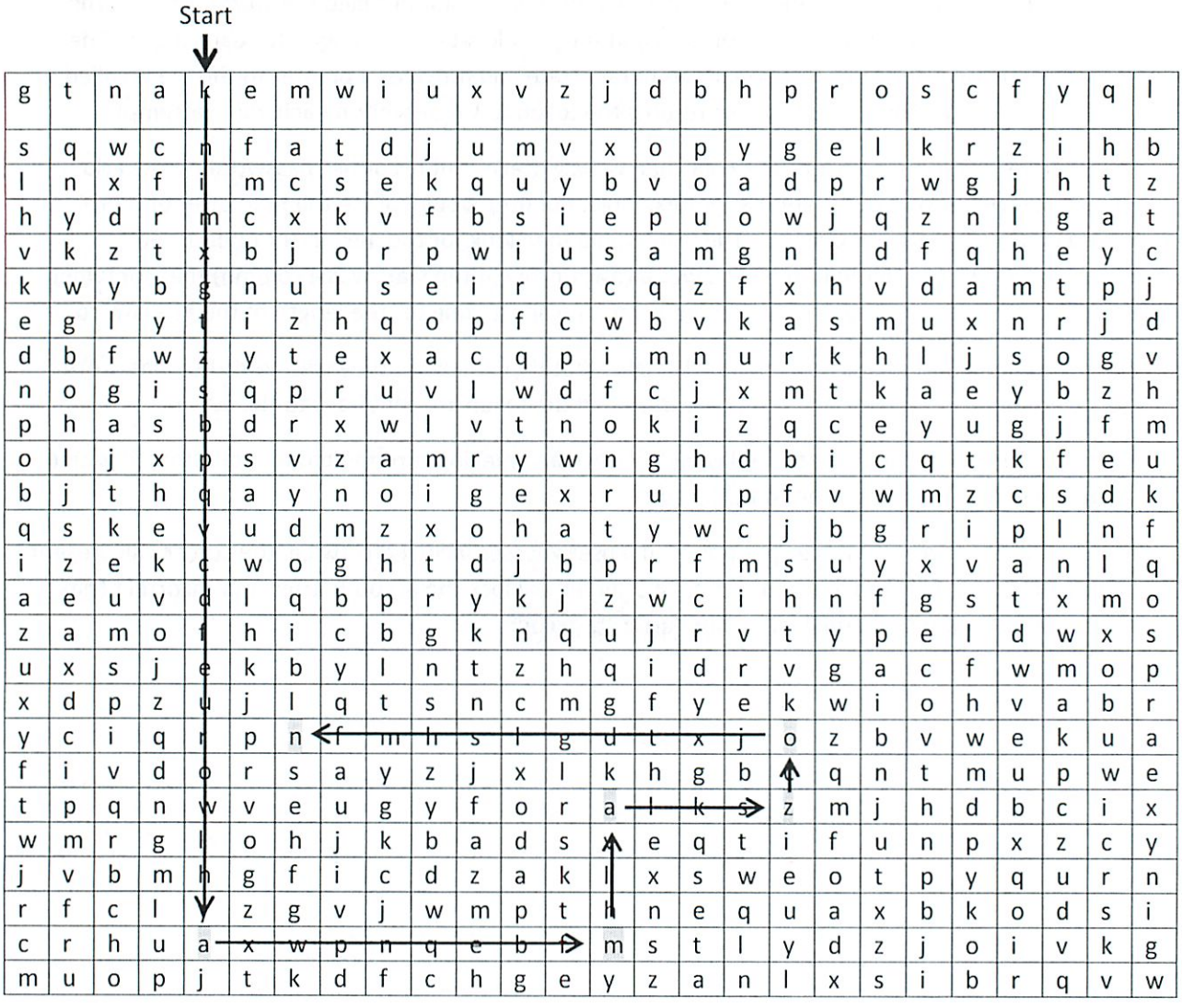
These are transmitted to the user. The user then visually traces out his or her password on the grid, alternating between rows and columns. For example, the user would locate the first letter of their password on the start row or column. The user would then look for the next letter of his or her password in either the column (if the start was a row) or row (if the start was a column) that contained the user's first character. The user would then continue alternating for the length of their password.

The user enters the directions (up, down, left, right) that they follow as they trace out their password. This is called the *trace* of the password. The trace and the token are sent back to the server.

The server verifies that the trace by replaying the trace and making sure the password letters fit within the code. Changed the description to fit the code

The server will only accept 2 traces per token. If a user guesses incorrectly twice, the server will present the user with a new Grid and Start Location. The server will lock the account after four incorrect tries until the user completes an email loop. not currently in code

Example: entering the password Amazon with the 5th column as the start row/column. The grid as well as the start row/column are randomly generated be the server for each log in.



g	t	n	a	k	e	m	w	i	u	x	v	z	j	d	b	h	p	r	o	s	c	f	y	q	l
s	q	w	c	m	f	a	t	d	j	u	m	v	x	o	p	y	g	e	l	k	r	z	i	h	b
l	n	x	f	i	m	c	s	e	k	q	u	y	b	v	o	a	d	p	r	w	g	j	h	t	z
h	y	d	r	m	c	x	k	v	f	b	s	i	e	p	u	o	w	j	q	z	n	l	g	a	t
v	k	z	t	x	b	j	o	r	p	w	i	u	s	a	m	g	n	l	d	f	q	h	e	y	c
k	w	y	b	g	n	u	l	s	e	i	r	o	c	q	z	f	x	h	v	d	a	m	t	p	j
e	g	l	y	t	i	z	h	q	o	p	f	c	w	b	v	k	a	s	m	u	x	n	r	j	d
d	b	f	w	z	y	t	e	x	a	c	q	p	i	m	n	u	r	k	h	l	j	s	o	g	v
n	o	g	i	s	q	p	r	u	v	l	w	d	f	c	j	x	m	t	k	a	e	y	b	z	h
p	h	a	s	b	d	r	x	w	l	v	t	n	o	k	i	z	q	c	e	y	u	g	j	f	m
o	l	j	x	p	s	v	z	a	m	r	y	w	n	g	h	d	b	i	c	q	t	k	f	e	u
b	j	t	h	q	a	y	n	o	i	g	e	x	r	u	l	p	f	v	w	m	z	c	s	d	k
q	s	k	e	y	u	d	m	z	x	o	h	a	t	y	w	c	j	b	g	r	i	p	l	n	f
i	z	e	k	c	w	o	g	h	t	d	j	b	p	r	f	m	s	u	y	x	v	a	n	l	q
a	e	u	v	d	l	q	b	p	r	y	k	j	z	w	c	i	h	n	f	g	s	t	x	m	o
z	a	m	o	f	h	i	c	b	g	k	n	q	u	j	r	v	t	y	p	e	l	d	w	x	s
u	x	s	j	e	k	b	y	l	n	t	z	h	q	i	d	r	v	g	a	c	f	w	m	o	p
x	d	p	z	u	j	l	q	t	s	n	c	m	g	f	y	e	k	w	i	o	h	v	a	b	r
y	c	i	q	r	p	n	f	m	h	s	l	g	d	t	x	j	o	z	b	v	w	e	k	u	a
f	i	v	d	o	r	s	a	y	z	j	x	l	k	h	g	b	q	n	t	m	u	p	w	e
t	p	q	n	w	v	e	u	g	y	f	o	r	a	l	k	s	z	m	j	h	d	b	c	i	x
w	m	r	g	l	o	h	j	k	b	a	d	s	x	e	q	t	i	f	u	n	p	x	z	c	y
j	v	b	m	h	g	f	i	c	d	z	a	k	l	x	s	w	e	o	t	p	y	q	u	r	n
r	f	c	l	y	z	g	v	j	w	m	p	t	h	n	e	q	u	a	x	b	k	o	d	s	i
c	r	h	u	a	x	w	p	n	q	e	b	i	m	s	t	l	y	d	z	j	o	i	v	k	g
m	u	o	p	j	t	k	d	f	c	h	g	e	y	z	a	n	l	x	s	i	b	r	q	v	w

The resulting trace would be: Down, Right, Up, Right, Up, Left.

Figure 9 A trace of the password "amazon"

Modified Tracer

We also explored a modified version of this system designed to increase usability. This system uses a 13x13 grid, instead of a 26x26 grid to make it easier for users to visually scan the grid.

In addition, we no longer generate a Latin Square. Instead, we first randomly distribute the user's password in an empty grid. We first randomly select either a row or a column from our 26 choices. We then place the first letter somewhere in that row or column. For this example, say we select the 3rd column to start with. We then place the "a" somewhere in this first column. We then place the second

letter "m" in the row in which we have placed the first letter. We continue this scheme until the password has been placed.

This scheme seems different from the code; it seems what I've written seems to have the same outcome but is simpler to explain and build. But the end result seems the same (or at least very similar). Are we sure it is the same from a probability perspective?

For example:

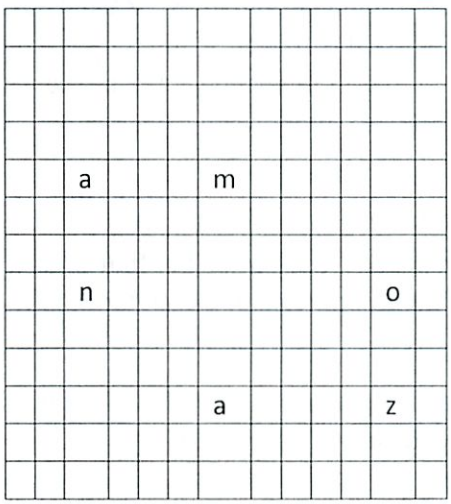


Figure 10

We then randomly fill in the remaining letters on the grid from the set of 26 lower case letters. We make sure each row and column only contains each letter only once by backtracking. For each spot we first start with the entire set [a...z]. We then remove all letters that are currently in the same row and column that we are in. We then randomly select a character from the remaining set.

Currently not built in code

This is not a Latin Square because we have a 13x13 Grid, but 26 possible characters.

Probabilistic Analysis (Miguel)

Original Off the Grid

Tracer: Trace the Grid

Modified Tracer

Math from different from no longer being a Latin Square.

Other Factors (Plaz)

We evaluate each system according to the criteria set out in The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes.[3]

Improvements to Criteria

Resilient-to-Physical Observations Category

We think that the **Resilient-to-Physical Observations** category should be split in two: **casual observation** and **video observation**. Casual observation is if an attacker is just able to watch the user enter their password once. This is feasible for short passwords and/or if the user types slow. An attacker can see which keys are hit on the keyboard. This is especially true if the user types slowly, has a short, and/or easily remember-able password.

However, the attacker seeing the user trace out the password on the grid once would have trouble remembering the entire grid, preventing the total loss of the password scheme. For that specific domain name, the attacker would have trouble remembering the sequence of 12 random characters, providing some additional security.

Video observation is defined as the attacker having the full ability to carefully watch and study users' movements because the attacker is able to pause and replay the user's log in actions.

Resilient-to-Internal Observation Category

We propose breaking up the **Resilient-to-Internal Observation** category into: **internal observation** and **wire observation**. We feel that splitting these categories allows us to elaborate on the strength of the designs.

Must Seek Out

Must the user seek out the new password system? Or does the server require that the user use it?

Man in the middle?

Original Off the Grid

Usability benefits

1. **Memorywise-Effortless YES** There are no secrets to be remembered in the base case. The description mentions a more advanced case, where the user could start at a different location, but we are assuming the base case where the user automatically selects the same location.
2. **Scalable-for-Users YES** The user only needs one grid for all of their sites.
3. **Nothing-to-Carry NO** User must carry 1 sheet of paper
4. **Physically-Effortless NO** The user must trace out their password on paper
5. **Easy-to-Learn NO** Using the same rubric as the paper does, the scheme is quite complicated
6. **Efficient-to-Use NO** The scheme requires a fair amount of effort for each authentication.
7. **Infrequent-Errors NO** Tracing out the password on the grid twice is easy to mess up.

[3] http://css.csail.mit.edu/6.858/2012/readings/passwords.pdf

8. **Easy-Recovery-from-Loss KINDA** If the user lost their Grid, they must have another copy of their Grid, or the key used to generate that Grid. A user can always reset their passwords on each site. The paper rates generic passwords as **Easy-Recovery-from-Loss YES**.

Deployability benefit

1. **Accessible KINDA** The user needs to be sighted to use this scheme. There could conceivably be a braille-based grid, but not at this moment.
2. **Negligible-Cost-per-User YES** The user is required to print one sheet of paper which costs < 05 cents
3. **Server-Compatibility YES** One of the primary benefits of this scheme is that it is compatible with existing servers which use passwords
4. **Browser-Compatibility YES** No special browser is needed
5. **Mature KINDA** The scheme has been published for some length of time; at least one Android app exists with support.
6. **Non-Proprietary YES** The scheme is published fully.

Security benefits

1. **Resilient-to-Physical Observations-Casual KINDA** The attacker would have to remember 12 random characters in order to observe the user's password for that site.
   With just a casual observation there is no way the attacker can memorize the entire Grid.
2. **Resilient-to-Physical Observations-Video NO** If the attacker can take a picture of the Grid, for example, a video camera over the shoulder, then the attacker would have access to all of the users' passwords assuming the user is using the standard Of the Grid scheme.
3. **Resilient-to-Targeted-Impersonation YES** Personal knowledge cannot help for the Off the Grid scheme. However, the normal password recovery mechanisms of the website remain, which are generally very vulnerable to Targeted Impersonation.
4. **Resilient-to-Throttled-Guessing YES** The user's password is 12 random alphanumeric characters. This means there are 26^{12} possible passwords.
5. **Resilient-to-Unthrottled-Guessing YES** There are 26^{12} possible passwords.
6. **Resilient-to-Internal-Observation NO** Off the Grid reduces to a normal 12 character password unique for each domain. This password is the same for each log in.
7. **Resilient-to-Wire-Observation KINDA** The password is the same for each log in; it must be protected with some additional protection (such as SSL) in transit.
8. **Resilient-to-Leaks-from-Other-Verifiers YES** Ideally the server should be hashing the password. Regardless, each domain has a unique password so leaking one password does not give one feasible information about another domains' password.
9. **Resilient-to-Phishing NO** If the attacker is able to spoof the domain name of the site, then the user will follow the same trace on the grid, providing the attacker their password.
10. **Resilient-to-Theft NO!** If the attacker gets your grid, it's game over, assuming you are sticking to the base Off the Grid algorithm. The author suggests that you make small personal tweaks to the algorithm in order to add resilience to theft.
11. **No-Trusted-Third-Party YES** The third party provides the code to generate the grid. However, that code runs in JavaScript on your local computer, allowing you to verify that the code is actually generating a unique grid and is not sending a copy to the third party. One could also write ones' own implementation of the Grid generation scheme to be sure.

12. **Requires-Explicit-Consent YES** The user must trace their password on the grid and then enter it onto the computer.
13. **Unlinkable YES** Since each user's Grid is so different, there is no feasible way to link users using the same scheme.

Tracer: Trace the Grid

Goal: prevent from seeing over wire

Note all are for the actual log in experience. This analysis does not consider creating a password; the process of which is similar to traditional password schemes.

Usability benefits

1. **Memorywise-Effortless NO** The user must remember a password to use Tracer. Ideally, that password should be different between sites. Since we only allow lowercase alphabetic characters without repeating letters, we may prevent users from using the same password on a site running Tracer than the user uses on all of their sites.
2. **Scalable-for-Users NO** Ideally the user has a different password for each site
3. **Nothing-to-Carry YES** There is nothing to carry
4. **Physically-Effortless NO** The user must trace out their password on-screen
5. **Easy-to-Learn NO** Using the same rubric as the paper, the scheme is quite complicated
6. **Efficient-to-Use NO** The scheme requires a fair amount of effort for each authentication.
7. **Infrequent-Errors NO** Tracing out the password on screen is easy to mess up
8. **Easy-Recovery-from-Loss YES** Tracer falls back on the same recovery mechanisms as traditional password sites, which is rated YES in the paper.

Deployability benefit

1. **Accessible KINDA** A screen reader would be tedious to use.
2. **Negligible-Cost-per-User YES** There is no cost.
3. **Server-Compatibility NO** The server must be provisioned with a new authentication library.
4. **Browser-Compatibility YES** No special browser is needed
5. **Mature NO** We are proposing it here
6. **Non-Proprietary YES** The scheme is published fully.

Security benefits

1. **Resilient-to-Physical Observations-Casual POSSIBLY** If the attacker could see the screen and the keyboard they could not uncover the user's password, unless the user traces the password with their finger.
2. **Resilient-to-Physical Observations-Video POSSIBLY** Even with being able to study the user as they enter their password, the attacker would not be able to recover a user's password, unless the user traces the password with their finger. This is one of the major design goals of this system.
3. **Resilient-to-Targeted-Impersonation YES** Personal knowledge cannot help for the Off the Grid scheme. However, the normal password recovery mechanisms of the website remain, which are generally very vulnerable to Targeted Impersonation.

4. **Resilient-to-Throttled-Guessing YES** An attacker can only submit two tracers per grid/start location. After two tries, the server will issue a new grid. The user then gets two more tries at a trace submission before the account is locked until an email loop is performed.
5. **Resilient-to-Unthrottled-Guessing NO** TBD
6. **Resilient-to-Internal-Observation QUASI** This is the major design goal of this system. An attacker needs _____ observations of both the *grid*, *Start Location*, and *trace* in order to crack the password. Less if the user picks a dictionary word. Quasi since report says if we need 20-30.
7. **Resilient-to-Wire-Observation QUASI** This is the same as internal, except if additional transport level security is added (ie SSL/TLS).
8. **Resilient-to-Leaks-from-Other-Verifiers NO** The password is stored in plain text on the server in order for the server to verify the password. This is not good practice.
9. **Resilient-to-Phishing YES** An attacker with just one trace could not submit that trace to another server, because the grid is randomized each time.
10. **Resilient-to-Theft YES** There is nothing to steal
11. **No-Trusted-Third-Party YES** There are no 3rd parties involved
12. **Requires-Explicit-Consent YES** The user must trace their password on the computer and enter the trace.
13. **Unlinkable YES** Like passwords, this scheme is unlinkable.

Modified Tracer

The modified tracer is more E**fficient-to-Use** and has less errors (**Infrequent-Errors**), however at the cost of an decreased **Resilient-to-Physical Observations-Casual**, **Resilient-to-Physical Observations-Video**, **Resilient-to-Throttled-Guessing** , **Resilient-to-Unthrottled-Guessing**, **Resilient-to-Internal Observation**, and **Resilient-to-Phishing**. However, all criteria are still the same according to the rubric. This is because the rubric focuses on theoretical possibility, and not the degree of.

Comparison Table

	Of the Grid	Tracer	Modified Tracer
Memorywise-effortless	Yes	No	No
Scalable for users.	Yes	No	No
Nothing-to-carry	No	Yes	Yes
Physically-effortless	No	No	No
Easy-to-Learn	No	No	No
Efficient-to-Use	No	No	No (More)
Infrequent-Errors	No	No	No (More)
Easy-Recovery-from-Loss	Kinda	Yes	Yes
Accessible	Kinda	Kinda	Kinda
Negligible-Cost-per-User	Yes	Yes	Yes

Server-Compatibility	Yes	No	No
Browser-Compatibility	Yes	Yes	Yes
Mature	Kinda	No	No
Non-Proprietary	Yes	Yes	Yes
Resilient-to-Physical Observations-Casual	Kinda	Possibly	Possibly (Less)
Resilient-to-Physical Observations-Video	No	Possibly	Possibly (Less)
Resilient-to-Targeted-Impersonation	Yes	Yes	Yes
Resilient-to-Throttled-Guessing	Yes	Yes	Yes (Less)
Resilient-to-Unthrottled-Guessing	Yes	No	No (Less)
Resilient-to-Internal-Observation	No	Quasi	Quasi (Less)
Resilient-to-Wire-Observation	Kinda	Quasi	Quasi
Resilient-to-Leaks-from-Other-Verifiers	Yes	No	No
Resilient-to-Phishing	No	Yes	Yes (Less)
Resilient-to-Theft	No!	Yes	Yes
No-Trusted-Third-Party	Yes	Yes	Yes
Requires-Explicit-Consent	Yes	Yes	Yes
Unlinkable	Yes	Yes	Yes

Usability (Plaz)

The simpler a system is, the more it will be used.

This section attempts to usability of grid systems.[4] The three core tenants of usability are: learnability, efficiency, and safety.

Learnability

Discoverability

Tracer is more discoverable than Off the Grid because the website you are creating an account with can let you know that the website uses Tracer. It is inherently discoverable. Off the Grid requires that you hear about the system in some way. Websites can still advise you of the presence of Off the Grid, but the Off the Grid system, as currently designed and designated, is not inherently discoverable.

[4] Material from MIT's 6.813 User Interface classes by Prof. Rob Miller Spring 2012.

Training

Tracer can be taught to users when they pick their password for the site. For example, sites could show users a video of how to use Tracer. Sites could also provide an interactive training tool using Tracer that uses JavaScript and HTML 5 to show the user how to trace their actual password. Using the actual password would reveal the user's password to an attacker which can see a screen, but not a keyboard. Thus it should not be used in a public room. However, it would not change other security aspects of registration versus traditional password schemes because the password is still stored in the DOM.

Mental Model

We believe that once explained, it is easy to form a mental model of the system. The system asks you to solve a puzzle and you solve it. In addition, it is clear that this prevents you from sending your password over the wire for subsequent log ins. In addition, each log in is consistent with the rules of the system and ones' mental model of the system.

Efficiency

Tracer is more natural to use than Off the Grid because one can trace the system on the screen as one enters the keyboard traces. We feel that expert users of Tracer could use the arrow keys without taking their eyes off the screen. This could make password entry quite fast.

This is easier than Off the Grid which requires users to trace their password on the grid twice. Off the Grid also requires users to overshoot and enter two characters after each character in their password in Phase 2.

Chunking

Research has shown that people can remember 7 ±2 pieces of information at once.[5] Tracer requires that the user only remember one password.

Fitts's Law

Fitts's Law is an estimate of the time it takes someone to point to an object or steer among objects.[6] The rule as proposed by Scott MacKenzie is as follows:[7]

$$T = a + b \log_2(1 + \frac{D}{W})$$

where:

- T is the average time taken to complete the movement
- a represents reaction time to start moving
- b stands for the speed of movement
- D is the distance from the starting point to the center of the target.

---

[5] De Groot, A. D., *Thought and choice in chess*, 1965

[6] Paul M. Fitts (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, volume 47, number 6, June 1954, pp. 381–391.

[7] I. Scott MacKenzie and William A. S. Buxton (1992). Extending Fitts' law to two-dimensional tasks. Proceedings of ACM CHI 1992 Conference on Human Factors in Computing Systems, pp. 219–226. http://doi.acm.org/10.1145/142750.142794

- W is the width of the target measured along the axis of motion. W can also be thought of as the allowed error tolerance in the final position, since the final point of the motion must fall within $\pm W/2$ of the target's center.

We can use a more specific form to study steering tasks, the time to move your hand through a tunnel of length D and width S:

$$T = a + b\left(\frac{D}{S}\right)$$

The index of difficulty is now linear.

We can use this to measure the amount of time it takes someone to trace through the grid, assuming they trace the grid with their finger or mouse. Ideally the user should not do that to maintain **Resilient-to-Physical Observations-Casual** and **Resilient-to-Physical Observations-Video**.

Auto-Solver

It is possible for there to be a browser-based auto-solver for Tracer grids. This software would know the user's password and use that to automatically solve grid challenges. This would break **Resilient-to-Internal Observation** because the user's system would now need the password stored. However the system would still meet **Resilient-to-Wire-Observation** QUASI designation even without additional transport level security. It receives a Quasi designation because multiple successful log in attempts must be witnessed before the attacker has enough information to discover the password. It would do a great deal for usability, flipping **Physically-Effortless**, **Easy-to-Learn**, **Efficient-to-Use**, and **Infrequent-Errors** all to yes. In addition **Accessibility** would greatly improve.

Code (Jwang)

Anything we want to write here?

How do we generate a Latin Square?

Conclusion (?)

Tracer is not recommended

Draft 12/9

Probabilistic Analysis (Miguel) section

We evaluate the security of the three systems described above using a probability analysis. In eavesdropping on a user, an adversary may obtain the user's Square board, the password or trace, or both, the board and the password / trace. Given each of these pieces of information we see how the user may be compromised.

Original Off the Grid

The Off the Grid system offers users a unique password given the domain name. The user simply traces the domain name in the two phases described above.

The first attack may occur if an adversary obtains a user's grid. This is usually held on hand by the user, and may occur if their wallet is stolen.

→

- only correct ones

Can see Board/St Loc/Trace - how many attempts

Or what is data you are gathering ...?

g	e	a	m	o	n	z	k	i	r	c
k	a	n	c	m	z	o	r	g	i	e
n	k	c	z	a	m	r	g	o	e	i
z	o	i	a	n	g	e	c	r	k	m
m	r	z	n	g	a	k	i	e	c	o
a	g	e	i	z	r	n	o	c	m	k
r	c	g	k	e	i	m	n	z	o	a
e	n	k	g	i	o	c	z	m	a	r
i	m	o	e	r	c	g	a	k	n	k
c	i	r	o	k	e	a	m	n	z	g
o	z	m	r	c	k	i	e	a	g	n

Figure 9 Blank User's Grid

If an attacker is able to retrieve the grid of a user, he obtains no information about the user's password. If the attacker does not know of the Off the Grid system, the grid will offer no information nor clue to the user's password. If however, the attacker does know the Off the Grid protocol and that the user uses it, then all passwords are compromised because the attacker can then follow the two phases described in order to get the password for any site such as Amazon.com.

If an attacker instead obtains the user's password for a single site, only interactions with that website are compromised. In the description described above, the password "gaznegmacmzg" is obtained by following the two phases of the Off the Grid system for Amazon. If the attacker obtains "gaznegmacmzg," unless he knows which website it belongs to, is useless, otherwise he can use it to log in the user's account for Amazon.

However the attacker obtains no information regarding other passwords for other website domains. In order to do so, the user must guess the grid that created the password. Given the Off the Grid Implementation described, there are at least $\frac{(n!)^{2n}}{n^{n^2}}$ boards for an n-sized board, which leads to at least 9.337×10^{426} boards for $n = 26$. Given this, there is still little information obtained to gaining the password information for any other site.

Lastly, if an attacker retrieves the password and the grid, the user is equally as susceptible to the attacks described in (1). The adversary can now trace out the password in order to determine the website the password belongs to, and if he knows of the Off the Grid system, can retrieve the password for all other websites.

The safety of this implementation relies on how much the user secures the grid. If the grid is stolen, then all of the user's passwords are compromised.

Tracer: Trace the Grid

Our tracer implementation relies on the security of the Latin Square. Since each password must be made up of only lower case letters and no-repeating characters, the password can be searched using a 26×26 Latin Square as specified above, resulting in at least 9.337×10^{426} possible grids.

If the attacker gains access to the board the user sees along with the start location, the attacker never gains any information on the password of the user even after multiple board configurations are given. Each board will always contain all 26 letters, and no information is ever gained about the password.

If the attacker instead gains only access to the user's key-logs, thus obtaining their input, they can never retrieve the password. The password will be impossible to obtain from only getting "up, right, up, left, down" and so on. The only information gained is the length of the password which is 1-to-1 with the trace. Even with multiple traces, the password will be impossible to obtain without the trace. An adversary can then do a brute-force guess on the board because he knows the length of the password. A password presented as Down, Right, Up, Right, Up, Left we know will give us a password of length 6. Additionally, we know that the first element is given through the start-location on the board. We can use the fact that the direction changes each turn, thus Left and Right will always be followed by Up or Down. Thus this gives us a $1 \times 2 \times 2 \times 2 \times 2 \times 2 = 32$ possible combinations which we can then brute force. Thus after finding out the length of a password through the trace, we know there are then 2^{l-1} possibilities given, l, the length of the password. Brute force is reduced by the frequency of the board change.

Lastly, the most important case is when an attacker gains information to the board, the start-location, and the trace. To begin with,

Modified Tracer

While our Tracer implementation relies on the security of the Latin Square, the modified version is no longer a Latin Square having a reduced size, 13×13, but still using all 26 letters as possibilities.

Unlike our previous implementation, the adversary gains information looking at multiple boards over time. One board offers no information about what letters are in the password, but using multiple boards, an adversary may use the boards to determine which letters are in the password. Under some password configurations, one of the letters of the alphabet may not appear in the grid. If we continue onwards with this, we will eliminate all letters that are not in the password. It is then a matter of determining the length of the password and see what are plausible passwords given the letters.

Similar to our Original Tracer implementation, the adversary gains no information when multiple traces are presented other than the length of the password. They can then brute force the password knowing there are 2^{l-1} possible combinations.

Lastly, there is the possibility of having all information of the board, the start-location, and the trace.

Anand

Sunday, December 09, 2012
8:47 PM

change amazon I'm example 2 ✓
1/32
but only 4 tries
so 4/32 tries
(31/32)^4 ∈ 1 - that, right?
10% for specific user
but can scan password

brute force

11%
technically $\frac{31}{32} \cdot \frac{30}{31} \cdot \frac{31}{32} \cdot \frac{30}{31}$
So 12%
EV is

not Clean
but then edges
info theory
how many bits of info
5bita
log_2(26) etc
bound on information needed

later letters
original password hard
since little info
which is good if don't know password
but easy for then to get into your account without password
like xor password could just send 1 or 0
but easy to guess

(how did he describe again

try to get bounds
hard to solve explicitly
could use Mathematica
or sigmas
all locations , all things,
nested sums

no induction
need all the info
if 1 char password could not figure out, 0 password
they get no info, but log in

so is a 2 char like a 1 char password
very directly sacrificing info
complicated lookup table
write up 2 char password

modified
way different
now can just to the website n times
must limit times can attempt to log in
would never recommend tell them part of the password
at most once

can we do each pair only have 1 letter
then all 4 choices
could you construct such a grid?
if made 1 could always switch the letters

but how use them actually using grid

but can do character by character
between 4 and 5
can cross off letter
(better than freq analysis)
but here know it by sure
freq analysis much harder
do in addition?

can save grid
10 grids high probably
on avg 5 grids
might have 2 or 3 changes

DOS from lock ours
and guess passwords

challenge response protocol

grc show full 26x26
11x11 is sample

trade off some for others
but not equally weighted

Entropy

12/10

Measure of uncertainty

5 bits of entropy in 6 character password

Similar to dictionary words ~6 - 1.3

12/11

Presentation Planning

3 min

Jungi Regular Latin Sq by tonight
Working for presentation

Start w/ ~~what~~ motivation
Prove know something w/o actually typing
If someone is watching
Next time prove differently

Only Tracer / Cross Password
↑ go w/ it

Type in pw when others watching
Authenticate by showing password
~~How~~ What color is ~~the~~ tennis ball
- green
- round

(7)

Slope

Say it is a disaster?

Or just what we can do to fix?

Jiwang - Demo

Miguel - Conclusion

Plaz - intro

Each makes own slide

1 min for Qn A / transition

Or not

At end go back for Tenis Ball

45 Plaz

45-60 Miguel

(3)

? PDT demo instead ?

Yeah
easier to show

CrossPassword

theplaz, jwang7, mflores

6.858 Project

12/12/2012

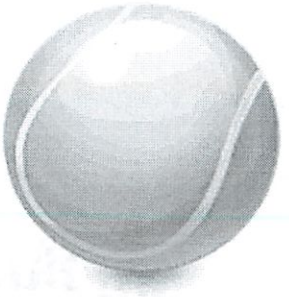
12/12 Presentation Slides

Someone can see you type it in

When you type in a password

Could you somehow prove you know the password without directly entering it?

Let's say your password is
tennis ball



What color is it?

green

What shape is it?

round

Could we enter those instead?

Introducing

CrossPassword

Demo

Password: **abc**

Let's say your password is
tennis ball

What color is it?

green

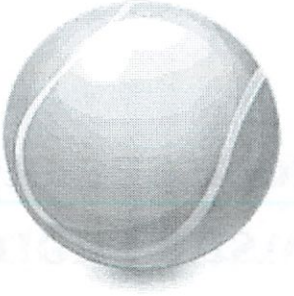
What shape is it?

round

Sound Familiar?

With enough questions answered, you can guess the password.

Let's say your password is
tennis ball



What color is it?

green **blue** **red** **black**
yellow **orange**

What shape is it?

round **rectangular**
triangular **none**

Limited answers can lead to brute force attacks.

CrossPassword: How to fix it?

- Increase bits of entropy in the response
- Rate limit password attempts
- Increase usability/decrease time to log in
  - Stripe the direction currently looking at

Many tradeoffs!

12/13
4P

CrossPassword: Novel Password Systems Where Enter Derivation of Password instead of Actual Password

6.858 Final Project

Michael Plasmeier <theplaz>
Jonathan Wang <jwang7>
Miguel Flores <mflores>

Motivation

The problem with many password systems is that users must type their entire, full password each time they log on. This makes the password vulnerable to key logging and interception during transmission.

We explore systems in which the user does not enter their direct password, but a derivation of the password **which changes on each log in**. The user proves that he or she knows the password without subsequently ever providing the password itself.

ING Password Keyboard

A simple example is ING Direct's PIN pad. Under ING's system, the user enters the letters corresponding to their PIN instead of the PIN itself. The mapping between numbers and letters is randomly generated on every log in. This method does not survive an attack where the attacker has access to the mapping, but it does prevent simple keylogging.

Figure 1 ING's Pin Pad. The user enters the letters corresponding to their PIN in the box.

Description of System (Plaz)

Original Off the Grid

We were inspired by the "Off the Grid" system from the Gibson Research Corporation.[1] The "Off the Grid" proposal is designed to allow users to use a personal printed paper grid to encipher the domain name of the website they are currently on into a string of psudeo-random characters.

The Off the Grid system works entirely on the user's side. Websites do not need to do anything to support Off the Grid.

To use Off the Grid, the user first generates a grid from a grid-providing website such as https://www.grc.com/offthegrid.htm. This website generates a grid using client-side scripting (ie. JavaScript) to generate the grid on the user's machine. The user then prints the grid onto a sheet of letter paper. At this point the Grid is offline and thus impossible to access by malware. As an alternative, there is at least one application for Android which produces and stores a grid; however, the grid is now accessible to malware on the Android phone which is able to defeat the inter-process sandboxing.

The grid that is generated is a Latin Square. A Latin Square is an $n \times n$ array filled with n different symbols, each occurring exactly once in each row and exactly once in each column.[2] The most famous Latin Square is the popular puzzle game Sudoku. (Note however, that we do not divide up the grid into 9 smaller 3x3 mini-squares in which each symbol must be unique). For example, here is a 11x11 Latin Square with 11 alphabetic characters:

g	e	a	m	o	n	z	k	i	r	c
k	a	n	c	m	z	o	r	g	i	e
n	k	c	z	a	m	r	g	o	e	i
z	o	i	a	n	g	e	c	r	k	m
m	r	z	n	g	a	k	i	e	c	o
a	g	e	i	z	r	n	o	c	m	k
r	c	g	k	e	i	m	n	z	o	a
e	n	k	g	i	o	c	z	m	a	r
i	m	o	e	r	c	g	a	k	n	k
c	i	r	o	k	e	a	m	n	z	g
o	z	m	r	c	k	i	e	a	g	n

Figure 2 An 11x11 Latin Square; normally 26x26, but reduced in size here to save space.

Once the user has a grid, they use the grid to create or change the password for each website. The Off the Grid specification has a number of variants, but we will use the base variant described on the GRC website.

In the Off the Grid specification, the user traces the name of the website twice to provide additional entropy. In the start of the first phase, the user always starts along the first row of the grid.

---

[1] https://www.grc.com/offthegrid.htm and associated pages. Is still marked as "Work in Progress;" Retrieved 12/2/2012

[2] http://en.wikipedia.org/wiki/Latin_square

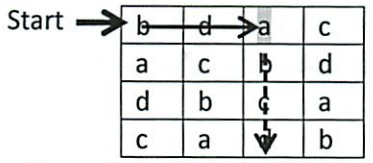


Figure 3

The user then traces out the first 6 characters of the domain name. 6 characters was chosen by the author to provide a 12 character password, which the author chose to balance ease of use with entropy. Again, a user may choose their own scheme. The user alternates between looking horizontally and vertically.

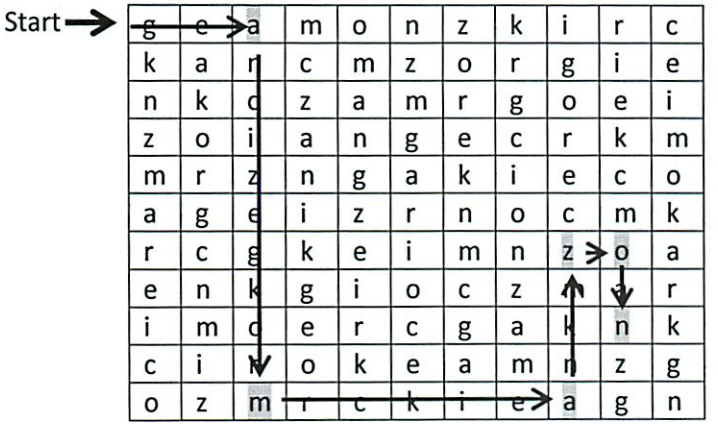


Figure 4

In the second phase, the user starts at the character that they ended with at the end of Phase 1. The user then two more characters from the grid in the same direction of travel. The user then appends those two characters to their password.

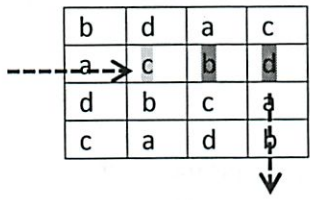


Figure 5 The user arrives at c traveling to the right. The user appends the next two characters "bd" to their password, and then continues up/down from the last character they read "d".

The user wraps around if their characters go off the grid.

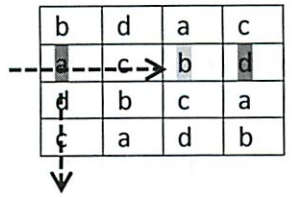


Figure 6 The user arrives at b traveling to the right. The user appends the next two characters to their password, wrapping around if they go off the edge of the grid. Here those characters are "da". The user then continues up/down from the last character "a".

For example here is Phase 2 of our Amazon example.

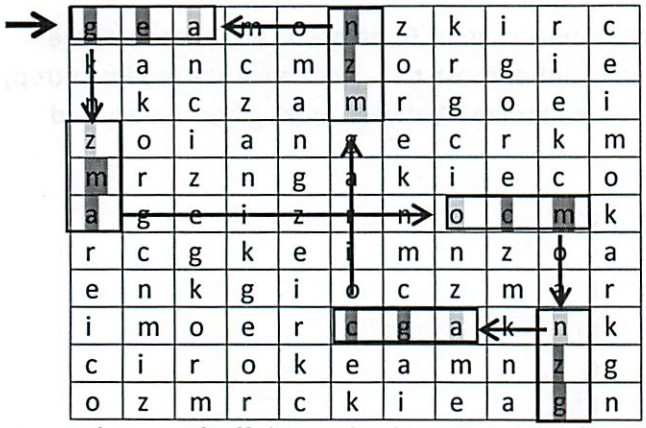


Figure 7 Phase 2 of Off the Grid. The password is "gaznegmacmzg"

Here are Phase 1 and Phase 2.

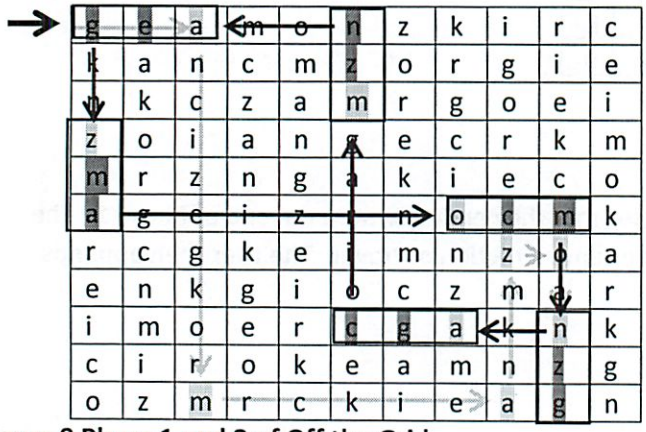


Figure 8 Phase 1 and 2 of Off the Grid.

To log in, the user retraces exactly the same steps as when creating a password. This means the password is exactly the same for each domain. This is an obvious requirement for a system designed to fit within the existing password infrastructure. However, we wanted to explore ideas in which the user does not enter the same password each time.

CrossPassword

We wanted to design a system similar to the Off the Grid system, but where the password the user transmits over the network is different each time. With this system, the website presents the user with a grid and the user enters only a deviation of their password.

When the user creates an account, he provides his or her password to the webserver. The user may use characters from the lower case Latin alphabet [a...z]. The password may not have consecutive repeating characters, for example, "aardvark". The password is stored on the server such that the plain text can be accessed in order to verify the trace.

When the user logs in, the server randomly generates a 26x26 Latin square with the characters [a...z] called the *Grid*. The server also randomly selects a start row or column called the *start location*. The server transmits this Grid to the user. The Grid and the start location are unique for each log in. The server stores the Grid and start location in temporary state and provides a pointer to this state called the *token* to the user. The user's browser returns the token to the server on each log in attempt.

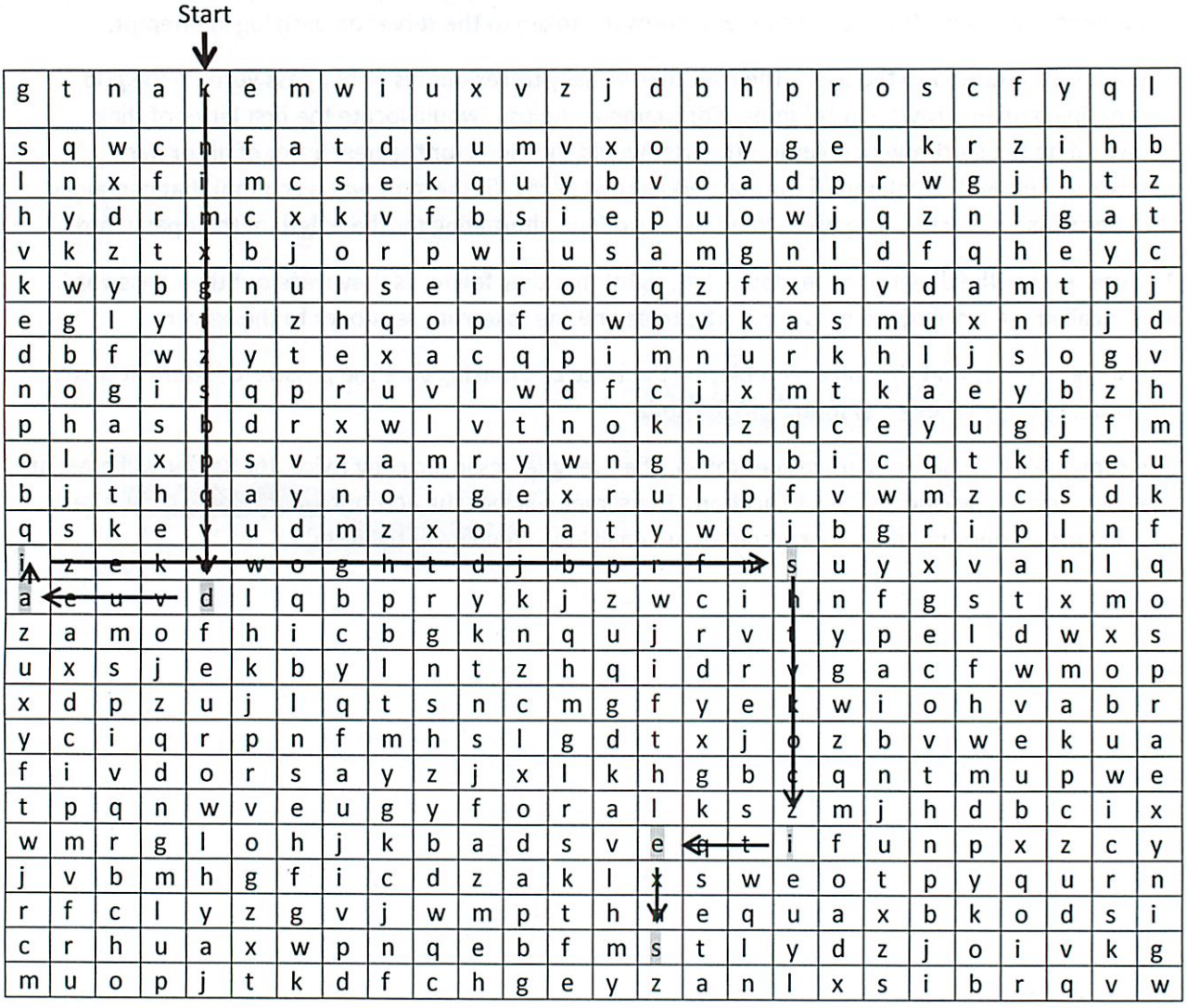
These are transmitted to the user. The user then visually traces out his or her password on the grid, alternating between rows and columns. For example, the user would locate the first letter of their password on the start row or column. The user would then look for the next letter of his or her password in either the column (if the start was a row) or row (if the start was a column) that contained the user's first character. The user would then continue alternating for the length of their password.

The user enters the directions (up, down, left, right) that they follow as they trace out their password. This is called the *trace* of the password. The trace and the token are sent back to the server.

The server verifies that the trace by replaying the trace and making sure the password letters fit within the code. Changed the description to fit the code

The server will only accept 2 traces per token. If a user guesses incorrectly twice, the server will present the user with a new Grid and Start Location. The server will lock the account and the IP address after four incorrect tries until the user completes an email loop. not currently in code

Example: entering the password Amazon with the 5th column as the start row/column. The grid as well as the start row/column are randomly generated be the server for each log in.



g	t	n	a	k	e	m	w	i	u	x	v	z	j	d	b	h	p	r	o	s	c	f	y	q	l
s	q	w	c	n	f	a	t	d	j	u	m	v	x	o	p	y	g	e	l	k	r	z	i	h	b
l	n	x	f	i	m	c	s	e	k	q	u	y	b	v	o	a	d	p	r	w	g	j	h	t	z
h	y	d	r	m	c	x	k	v	f	b	s	i	e	p	u	o	w	j	q	z	n	l	g	a	t
v	k	z	t	x	b	j	o	r	p	w	i	u	s	a	m	g	n	l	d	f	q	h	e	y	c
k	w	y	b	g	n	u	l	s	e	i	r	o	c	q	z	f	x	h	v	d	a	m	t	p	j
e	g	l	y	t	i	z	h	q	o	p	f	c	w	b	v	k	a	s	m	u	x	n	r	j	d
d	b	f	w	z	y	t	e	x	a	c	q	p	i	m	n	u	r	k	h	l	j	s	o	g	v
n	o	g	i	s	q	p	r	u	v	l	w	d	f	c	j	x	m	t	k	a	e	y	b	z	h
p	h	a	s	b	d	r	x	w	l	v	t	n	o	k	i	z	q	c	e	y	u	g	j	f	m
o	l	j	x	p	s	v	z	a	m	r	y	w	n	g	h	d	b	i	c	q	t	k	f	e	u
b	j	t	h	q	a	y	n	o	i	g	e	x	r	u	l	p	f	v	w	m	z	c	s	d	k
q	s	k	e	y	u	d	m	z	x	o	h	a	t	y	w	c	j	b	g	r	i	p	l	n	f
i	z	e	k	w	o	g	h	t	d	j	b	p	r	f	m	s	u	y	x	v	a	n	l	q	
a	e	u	v	d	l	q	b	p	r	y	k	j	z	w	c	i	h	n	f	g	s	t	x	m	o
z	a	m	o	f	h	i	c	b	g	k	n	q	u	j	r	v	t	y	p	e	l	d	w	x	s
u	x	s	j	e	k	b	y	l	n	t	z	h	q	i	d	r	y	g	a	c	f	w	m	o	p
x	d	p	z	u	j	l	q	t	s	n	c	m	g	f	y	e	k	w	i	o	h	v	a	b	r
y	c	i	q	r	p	n	f	m	h	s	l	g	d	t	x	j	o	z	b	v	w	e	k	u	a
f	i	v	d	o	r	s	a	y	z	j	x	l	k	h	g	b	c	q	n	t	m	u	p	w	e
t	p	q	n	w	v	e	u	g	y	f	o	r	a	l	k	s	z	m	j	h	d	b	c	i	x
w	m	r	g	l	o	h	j	k	b	a	d	s	v	e	q	t	i	f	u	n	p	x	z	c	y
j	v	b	m	h	g	f	i	c	d	z	a	k	l	x	s	w	e	o	t	p	y	q	u	r	n
r	f	c	l	y	z	g	v	j	w	m	p	t	h	w	e	q	u	a	x	b	k	o	d	s	i
c	r	h	u	a	x	w	p	n	q	e	b	f	m	s	t	l	y	d	z	j	o	i	v	k	g
m	u	o	p	j	t	k	d	f	c	h	g	e	y	z	a	n	l	x	s	i	b	r	q	v	w

The resulting trace would be: Down, Left, Up, Right, Down, Left, Down.

Figure 9 A trace of the password "daisies"

Modified CrossPassword

We also explored a modified version of this system designed to increase usability. This system uses a 13x13 grid, instead of a 26x26 grid to make it easier for users to visually scan the grid.

In addition, we no longer generate a Latin Square. Instead, we first randomly distribute the user's password in an empty grid. We first randomly select either a row or a column from our 26 choices. We then place the first letter somewhere in that row or column. For this example, say we select the 3rd column to start with. We then place the "a" somewhere in this first column. We then place the second

letter "m" in the row in which we have placed the first letter. We continue this scheme until the password has been placed.

This scheme seems different from the code; it seems what I've written seems to have the same outcome but is simpler to explain and build. But the end result seems the same (or at least very similar). Are we sure it is the same from a probability perspective?

For example:

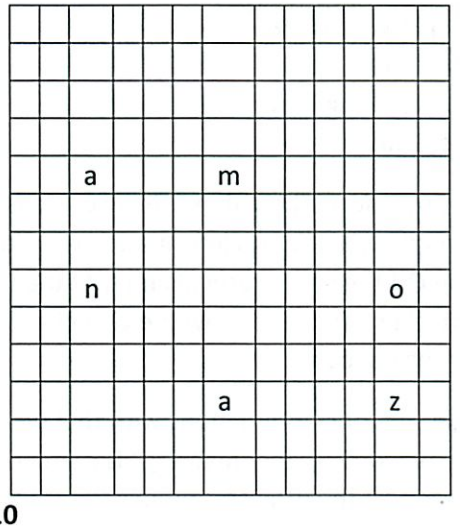


Figure 10

We then randomly fill in the remaining letters on the grid from the set of 26 lower case letters. We make sure each row and column only contains each letter only once by backtracking. For each spot we first start with the entire set [a...z]. We then remove all letters that are currently in the same row and column that we are in. We then randomly select a character from the remaining set. Currently not built in code this way

This is not a Latin Square because we have a 13x13 Grid, but 26 possible characters.

With this system we must prevent an attacker from looking at a grid a certain amount of times. A user can only look at up to 4 random grids.

Probabilistic Analysis (Miguel)

We evaluate the security of the three systems described above using a probability analysis. In eavesdropping on a user, an adversary may obtain the user's Square board, the password or trace, or both, the board and the password / trace. Given each of these pieces of information we see how the user may be compromised.

Original Off the Grid

The Off the Grid system offers users a unique password given the domain name. The user simply traces the domain name in the two phases described above.

The first attack may occur if an adversary obtains a user's grid. This is usually held on hand by the user, and may occur if their wallet is stolen.

g	e	a	m	o	n	z	k	i	r	c
k	a	n	c	m	z	o	r	g	i	e
n	k	c	z	a	m	r	g	o	e	i
z	o	i	a	n	g	e	c	r	k	m
m	r	z	n	g	a	k	i	e	c	o
a	g	e	i	z	r	n	o	c	m	k
r	c	g	k	e	i	m	n	z	o	a
e	n	k	g	i	o	c	z	m	a	r
i	m	o	e	r	c	g	a	k	n	k
c	i	r	o	k	e	a	m	n	z	g
o	z	m	r	c	k	i	e	a	g	n

Figure 9 Blank User's Grid

If an attacker is able to retrieve the grid of a user, he obtains no information about the user's password. If the attacker does not know of the Off the Grid system, the grid will offer no information nor clue to the user's password. If however, the attacker does know the Off the Grid protocol and that the user uses it, then all passwords are compromised because the attacker can then follow the two phases described in order to get the password for any site such as Amazon.com.

If an attacker instead obtains the user's password for a single site, only interactions with that website are compromised. In the description described above, the password "gaznegmacmzg" is obtained by following the two phases of the Off the Grid system for Amazon. If the attacker obtains "gaznegmacmzg," unless he knows which website it belongs to, is useless, otherwise he can use it to log in the user's account for Amazon.

However the attacker obtains no information regarding other passwords for other website domains. In order to do so, the user must guess the grid that created the password. Given the Off the Grid Implementation described, there are at least $\frac{(n!)^{2n}}{n^{n^2}}$ boards for an n-sized board, which leads to at least 9.337×10^{426} boards for $n = 26$. Given this, there is still little information obtained to gaining the password information for any other site.

Lastly, if an attacker retrieves the password and the grid, the user is equally as susceptible to the attacks described in (1). The adversary can now trace out the password in order to determine the website the password belongs to, and if he knows of the Off the Grid system, can retrieve the password for all other websites.

The safety of this implementation relies on how much the user secures the grid. If the grid is stolen, then all of the user's passwords are compromised.

CrossPassword

Our CrossPassword implementation relies on the security of the Latin Square. Since each password must be made up of only lower case letters and no-repeating characters, the password can be searched using a 26×26 Latin Square as specified above, resulting in at least 9.337×10^{426} possible grids.

If the attacker gains access to the board the user sees along with the start location, the attacker never gains any information on the password of the user even after multiple board configurations are given. Each board will always contain all 26 letters, and no information is ever gained about the password.

If the attacker instead gains only access to the user's key-logs, thus obtaining their input, they can never retrieve the password. The password will be impossible to obtain from only getting "up, right, up, left, down" and so on. The only information gained is the length of the password which is 1-to-1 with the trace. Even with multiple traces, the password will be impossible to obtain without the trace. An adversary can then do a brute-force guess on the board because he knows the length of the password. A password presented as Down, Right, Up, Right, Up, Left we know will give us a password of length 6. Additionally, we know that the first element is given through the start-location on the board. We can use the fact that the direction changes each turn, thus Left and Right will always be followed by Up or Down. Thus this gives us a $1 \times 2 \times 2 \times 2 \times 2 \times 2 = 32$ possible combinations which we can then brute force. Thus after finding out the length of a password through the trace, we know there are then 2^{l-1} possibilities given, l, the length of the password.

Brute force is reduced by the frequency of the board change and lockout. So there is a chance of guessing a specific user's password of

$$1 - \left(\frac{31}{32} * \frac{30}{31} * \frac{31}{32} * \frac{30}{31}\right) = 12\%$$

This means an attacker will likely figure out the password of about every ninth user if the attacker is able to use a range of IP addresses, since IP blocking.

Lastly, the most important case is when an attacker gains information to the board, the start-location, and the trace. To begin with,

Brute Force

I would add subheadings

Recover Password

However it is much more difficult for the attacker to backsolve the password back from the grid. Move content from above here

I would at least try to discuss how you know can't be in certain rows, to show that its difficult

Bound on information needed

2 character password example

Modified CrossPassword

While our CrossPassword implementation relies on the security of the Latin Square, the modified version is no longer a Latin Square having a reduced size, 13×13, but still using all 26 letters as possibilities.

Here the adversary does not even have to attempt to log in; they can just look at a number of boards

Determine by elimination which character is never missing (and plus frequency analysis as a bonus)

Use past boards; go letter by letter

In 10 refreshes, chance of getting the password is (calculation)

Unlike our previous implementation, the adversary gains information looking at multiple boards over time. One board offers no information about what letters are in the password, but using multiple boards, an adversary may use the boards to determine which letters are in the password. Under some password configurations, one of the letters of the alphabet may not appear in the grid. If we continue onwards with this, we will eliminate all letters that are not in the password. It is then a matter of determining the length of the password and see what are plausible passwords given the letters.

Similar to our Original CrossPassword implementation, the adversary gains no information when multiple traces are presented other than the length of the password. They can then brute force the password knowing there are 2^{l-1} possible combinations.

Lastly, there is the possibility of having all information of the board, the start-location, and the trace.

Why only 4 random grids?

Other Factors (Plaz)

We evaluate each system according to the criteria set out in The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes.[3]

Improvements to Criteria

Resilient-to-Physical Observations Category

We think that the **Resilient-to-Physical Observations** category should be split in two: **casual observation** and **video observation**. Casual observation is if an attacker is just able to watch the user enter their password once. This is feasible for short passwords and/or if the user types slow. An attacker can see which keys are hit on the keyboard. This is especially true if the user types slowly, has a short, and/or easily remember-able password.

However, the attacker seeing the user trace out the password on the grid once would have trouble remembering the entire grid, preventing the total loss of the password scheme. For that specific domain name, the attacker would have trouble remembering the sequence of 12 random characters, providing some additional security.

Video observation is defined as the attacker having the full ability to carefully watch and study users' movements because the attacker is able to pause and replay the user's log in actions.

Resilient-to-Internal Observation Category

We propose breaking up the **Resilient-to-Internal Observation** category into: **internal observation** and **wire observation**. We feel that splitting these categories allows us to elaborate on the strength of the designs.

Must Seek Out

Must the user seek out the new password system? Or does the server require that the user use it?

[3] http://css.csail.mit.edu/6.858/2012/readings/passwords.pdf

Man in the middle?

Denial of Service-able

An active attacker can cause a denial-of-service attack by submitting a sufficient quantity of incorrect passwords such that the system locks the user out.

Original Off the Grid

Usability benefits

1. **Memorywise-Effortless YES** There are no secrets to be remembered in the base case. The description mentions a more advanced case, where the user could start at a different location, but we are assuming the base case where the user automatically selects the same location.
2. **Scalable-for-Users YES** The user only needs one grid for all of their sites.
3. **Nothing-to-Carry NO** User must carry 1 sheet of paper
4. **Physically-Effortless NO** The user must trace out their password on paper
5. **Easy-to-Learn NO** Using the same rubric as the paper does, the scheme is quite complicated
6. **Efficient-to-Use NO** The scheme requires a fair amount of effort for each authentication.
7. **Infrequent-Errors NO** Tracing out the password on the grid twice is easy to mess up.
8. **Easy-Recovery-from-Loss KINDA** If the user lost their Grid, they must have another copy of their Grid, or the key used to generate that Grid. A user can always reset their passwords on each site. The paper rates generic passwords as **Easy-Recovery-from-Loss YES**.

Deployability benefit

1. **Accessible KINDA** The user needs to be sighted to use this scheme. There could conceivably be a braille-based grid, but not at this moment.
2. **Negligible-Cost-per-User YES** The user is required to print one sheet of paper which costs < 05 cents
3. **Server-Compatibility YES** One of the primary benefits of this scheme is that it is compatible with existing servers which use passwords
4. **Browser-Compatibility YES** No special browser is needed
5. **Mature KINDA** The scheme has been published for some length of time; at least one Android app exists with support.
6. **Non-Proprietary YES** The scheme is published fully.

Security benefits

1. **Resilient-to-Physical Observations-Casual KINDA** The attacker would have to remember 12 random characters in order to observe the user's password for that site.
   With just a casual observation there is no way the attacker can memorize the entire Grid.
2. **Resilient-to-Physical Observations-Video NO** If the attacker can take a picture of the Grid, for example, a video camera over the shoulder, then the attacker would have access to all of the users' passwords assuming the user is using the standard Of the Grid scheme.
3. **Resilient-to-Targeted-Impersonation YES** Personal knowledge cannot help for the Off the Grid scheme. However, the normal password recovery mechanisms of the website remain, which are generally very vulnerable to Targeted Impersonation.

4. **Resilient-to-Throttled-Guessing YES** The user's password is 12 random alphanumeric characters. This means there are 26^{12} possible passwords.
5. **Resilient-to-Unthrottled-Guessing YES** There are 26^{12} possible passwords.
6. **Resilient-to-Internal-Observation NO** Off the Grid reduces to a normal 12 character password unique for each domain. This password is the same for each log in.
7. **Resilient-to-Wire-Observation KINDA** The password is the same for each log in; it must be protected with some additional protection (such as SSL) in transit.
8. **Resilient-to-Leaks-from-Other-Verifiers YES** Ideally the server should be hashing the password. Regardless, each domain has a unique password so leaking one password does not give one feasible information about another domains' password.
9. **Resilient-to-Phishing NO** If the attacker is able to spoof the domain name of the site, then the user will follow the same trace on the grid, providing the attacker their password.
10. **Resilient-to-Theft NO!** If the attacker gets your grid, it's game over, assuming you are sticking to the base Off the Grid algorithm. The author suggests that you make small personal tweaks to the algorithm in order to add resilience to theft.
11. **No-Trusted-Third-Party YES** The third party provides the code to generate the grid. However, that code runs in JavaScript on your local computer, allowing you to verify that the code is actually generating a unique grid and is not sending a copy to the third party. One could also write ones' own implementation of the Grid generation scheme to be sure.
12. **Requires-Explicit-Consent YES** The user must trace their password on the grid and then enter it onto the computer.
13. **Unlinkable YES** Since each user's Grid is so different, there is no feasible way to link users using the same scheme.
14. **Denial-of-Service-able NO** This is the same as normal passwords. Under a normal password system, services generally do not add a lockout provision.

CrossPassword

Goal: prevent from seeing over wire

Note all are for the actual log in experience. This analysis does not consider creating a password; the process of which is similar to traditional password schemes.

Usability benefits

1. **Memorywise-Effortless NO** The user must remember a password to use CrossPassword. Ideally, that password should be different between sites. Since we only allow lowercase alphabetic characters without repeating letters, we may prevent users from using the same password on a site running CrossPassword than the user uses on all of their sites.
2. **Scalable-for-Users NO** Ideally the user has a different password for each site
3. **Nothing-to-Carry YES** There is nothing to carry
4. **Physically-Effortless NO** The user must trace out their password on-screen
5. **Easy-to-Learn NO** Using the same rubric as the paper, the scheme is quite complicated
6. **Efficient-to-Use NO** The scheme requires a fair amount of effort for each authentication.
7. **Infrequent-Errors NO** Tracing out the password on screen is easy to mess up
8. **Easy-Recovery-from-Loss YES** CrossPassword falls back on the same recovery mechanisms as traditional password sites, which is rated YES in the paper.

Deployability benefit

1. **Accessible KINDA** A screen reader would be tedious to use.
2. **Negligible-Cost-per-User YES** There is no cost.
3. **Server-Compatibility NO** The server must be provisioned with a new authentication library.
4. **Browser-Compatibility YES** No special browser is needed
5. **Mature NO** We are proposing it here
6. **Non-Proprietary YES** The scheme is published fully.

Security benefits

1. **Resilient-to-Physical Observations-Casual POSSIBLY** If the attacker could see the screen and the keyboard they could not uncover the user's password, unless the user traces the password with their finger.
2. **Resilient-to-Physical Observations-Video POSSIBLY** Even with being able to study the user as they enter their password, the attacker would not be able to recover a user's password, unless the user traces the password with their finger. This is one of the major design goals of this system.
3. **Resilient-to-Targeted-Impersonation YES** Personal knowledge cannot help for the Off the Grid scheme. However, the normal password recovery mechanisms of the website remain, which are generally very vulnerable to Targeted Impersonation.
4. **Resilient-to-Throttled-Guessing YES** An attacker can only submit two tracers per grid/start location. After two tries, the server will issue a new grid. The user then gets two more tries at a trace submission before the account is locked until an email loop is performed.
5. **Resilient-to-Unthrottled-Guessing NO** TBD
6. **Resilient-to-Internal-Observation QUASI** This is the major design goal of this system. An attacker needs _____ observations of both the *grid*, *Start Location*, and *trace* in order to crack the password. Less if the user picks a dictionary word. Quasi since report says if we need 20-30.
7. **Resilient-to-Wire-Observation QUASI** This is the same as internal, except if additional transport level security is added (ie SSL/TLS).
8. **Resilient-to-Leaks-from-Other-Verifiers NO** The password is stored in plain text on the server in order for the server to verify the password. This is not good practice.
9. **Resilient-to-Phishing YES** An attacker with just one trace could not submit that trace to another server, because the grid is randomized each time.
10. **Resilient-to-Theft YES** There is nothing to steal
11. **No-Trusted-Third-Party YES** There are no 3rd parties involved
12. **Requires-Explicit-Consent YES** The user must trace their password on the computer and enter the trace.
13. **Unlinkable YES** Like passwords, this scheme is unlinkable.
14. **Denial-of-Service-able YES** An attacker can lock out an account by trying an incorrect password 4 times.

Modified CrossPassword

The modified CrossPassword is more **Efficient-to-Use** and has less errors (**Infrequent-Errors**), however at the cost of an decreased **Resilient-to-Physical Observations-Casual, Resilient-to-Physical Observations-Video, Resilient-to-Throttled-Guessing , Resilient-to-Unthrottled-Guessing, Resilient-to-**

Internal Observation, and **Resilient-to-Phishing**. However, all criteria are still the same according to the rubric. This is because the rubric focuses on theoretical possibility, and not the degree of.

Modified CrossPassword is even more **Denial-of-Service-able**. An attacker can now launch a Denial of Service attack simply by displaying the grid, so we need to put in place a more aggressive lock out policy which restricts not only the number of guesses, but the number of grids which are displayed.

Comparison Table

	Of the Grid	CrossPass word	Modified CrossPass word
Memorywise-effortless	Yes	No	No
Scalable for users.	Yes	No	No
Nothing-to-carry	No	Yes	Yes
Physically-effortless	No	No	No
Easy-to-Learn	No	No	No
Efficient-to-Use	No	No	No (More)
Infrequent-Errors	No	No	No (More)
Easy-Recovery-from-Loss	Kinda	Yes	Yes
Accessible	Kinda	Kinda	Kinda
Negligible-Cost-per-User	Yes	Yes	Yes
Server-Compatibility	Yes	No	No
Browser-Compatibility	Yes	Yes	Yes
Mature	Kinda	No	No
Non-Proprietary	Yes	Yes	Yes
Resilient-to-Physical Observations-Casual	Kinda	Possibly	Possibly (Less)
Resilient-to-Physical Observations-Video	No	Possibly	Possibly (Less)
Resilient-to-Targeted-Impersonation	Yes	Yes	Yes
Resilient-to-Throttled-Guessing	Yes	Yes	Yes (Less)
Resilient-to-Unthrottled-Guessing	Yes	No	No (Less)
Resilient-to-Internal-Observation	No	Quasi	Quasi (Less)
Resilient-to-Wire-Observation	Kinda	Quasi	Quasi
Resilient-to-Leaks-from-Other-Verifiers	Yes	No	No
Resilient-to-Phishing	No	Yes	Yes (Less)

Resilient-to-Theft	No!	Yes	Yes
No-Trusted-Third-Party	Yes	Yes	Yes
Requires-Explicit-Consent	Yes	Yes	Yes
Unlinkable	Yes	Yes	Yes
Denial-of-Service-able	No	Yes	Yes (More)

Usability (Plaz)

The simpler a system is, the more it will be used.

This section attempts to usability of grid systems.[4] The three core tenants of usability are: learnability, efficiency, and safety.

Learnability

Discoverability

CrossPassword is more discoverable than Off the Grid because the website you are creating an account with can let you know that the website uses CrossPassword. It is inherently discoverable. Off the Grid requires that you hear about the system in some way. Websites can still advise you of the presence of Off the Grid, but the Off the Grid system, as currently designed and designated, is not inherently discoverable.

Training

CrossPassword can be taught to users when they pick their password for the site. For example, sites could show users a video of how to use CrossPassword. Sites could also provide an interactive training tool using CrossPassword that uses JavaScript and HTML 5 to show the user how to trace their actual password. Using the actual password would reveal the user's password to an attacker which can see a screen, but not a keyboard. Thus it should not be used in a public room. However, it would not change other security aspects of registration versus traditional password schemes because the password is still stored in the DOM.

Mental Model

We believe that once explained, it is easy to form a mental model of the system. The system asks you to solve a puzzle and you solve it. In addition, it is clear that this prevents you from sending your password over the wire for subsequent log ins. In addition, each log in is consistent with the rules of the system and ones' mental model of the system.

Efficiency

CrossPassword is more natural to use than Off the Grid because one can trace the system on the screen as one enters the keyboard traces. We feel that expert users of CrossPassword could use the arrow keys without taking their eyes off the screen. This could make password entry quite fast.

[4] Material from MIT's 6.813 User Interface classes by Prof. Rob Miller Spring 2012.

This is easier than Off the Grid which requires users to trace their password on the grid twice. Off the Grid also requires users to overshoot and enter two characters after each character in their password in Phase 2.

Chunking

Research has shown that people can remember 7 ±2 pieces of information at once.[5] CrossPassword requires that the user only remember one password.

Casual observation; 12 not possible

Fitts's Law

Fitts's Law is an estimate of the time it takes someone to point to an object or steer among objects.[6] The rule as proposed by Scott MacKenzie is as follows:[7]

$$T = a + b \log_2(1 + \frac{D}{W})$$

where:

- T is the average time taken to complete the movement
- a represents reaction time to start moving
- b stands for the speed of movement
- D is the distance from the starting point to the center of the target.
- W is the width of the target measured along the axis of motion. W can also be thought of as the allowed error tolerance in the final position, since the final point of the motion must fall within $\pm W/2$ of the target's center.

We can use a more specific form to study steering tasks, the time to move your hand through a tunnel of length D and width S:

$$T = a + b (\frac{D}{S})$$

The index of difficulty is now linear.

We can use this to measure the amount of time it takes someone to trace through the grid, assuming they trace the grid with their finger or mouse. Ideally the user should not do that to maintain **Resilient-to-Physical Observations-Casual** and **Resilient-to-Physical Observations-Video**.

---

[5] De Groot, A. D., *Thought and choice in chess*, 1965

[6] Paul M. Fitts (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, volume 47, number 6, June 1954, pp. 381–391.

[7] I. Scott MacKenzie and William A. S. Buxton (1992). Extending Fitts' law to two-dimensional tasks. Proceedings of ACM CHI 1992 Conference on Human Factors in Computing Systems, pp. 219–226. http://doi.acm.org/10.1145/142750.142794

Auto-Solver

It is possible for there to be a browser-based auto-solver for CrossPassword grids. This software would know the user's password and use that to automatically solve grid challenges. This would break **Resilient-to-Internal Observation** because the user's system would now need the password stored. However the system would still meet **Resilient-to-Wire-Observation** QUASI designation even without additional transport level security. It receives a Quasi designation because multiple successful log in attempts must be witnessed before the attacker has enough information to discover the password. It would do a great deal for usability, flipping **Physically-Effortless**, **Easy-to-Learn**, **Efficient-to-Use**, and **Infrequent-Errors** all to yes. In addition **Accessibility** would greatly improve.

Code (Jwang)

Anything we want to write here?

How do we generate a Latin Square?

Conclusion (Plaz)

CrossPassword is not recommended as a password system. Modified CrossPassword turned out to be even weaker than we first imagined.

We controlled for the wrong thing. The password had a lot of information. However, the item which we return to the server has very little information. The **Shannon entropy** of CrossPassword is $(l - 1)$ where l is the number of characters in the password. This makes it easy to brute force.

For example, say we take a password and XOR the characters together to get 1 bit which is either yes or no. We transmit very little meaningful information of the password, but that very fact makes it easy for the attacker to guess!

We showed that trading off some factors in The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes produces different outcomes in security. Factors cannot be traded off one-for-one. The factors are not evenly weighted.

CrossPassword

12/13
4:30P

Ok finish off paper today

Fix intro conclusion
Fix ~~probability~~ notes in code
Fix UI section ← biggest job

Hard to fully review w/o code
Or the other guys having made their changes...

'ds man in middle
L w/o SSL
just to look at

---

add - discoverable - remove wide dfs
- SSL proxy
- choose any Pw
- denial of service

①

Oh added trace

Now wrap up on the other stuff...

Does anything change since don't need to submit a trace?

---

does modified need trace

└ needs 1 or 2

and a ~~bunch~~ ~10 grids

guessing is same...

So with a bunch of grids of original

└ case had to recover password

as Anand made clear!

Can get characters w/o trace

└ yeah doesn't need a trace!

(3)

12/13
7:40p

Visibility

; How do we want to structure?
; Why is this even a separate section
  L prof mentioned it would be good

We could say more about the steering
Exactly how long it would take someone to find a password...

Or present it as learnings from 6.813
Or a general section on visibility research

; What was that UI timing thing?
Or we could put a worst case bound
  but this is tracking w/ finger
We support eye - w/ shutting

(4)

Should think more about Fitts' Law
and 0 knowledge proofs

(reviewed 0 knowledge)

Now how does that apply here?

Same ~~goke~~ goal

Prover ~~has statement~~ (use)
wants to show knows pw to verifier

Verifier = server

So verifier asks qu
prover can always get it correct ✓
if it knows
if it doesn't - might get correct ✓
but we only have 1

(5)

Fitt's Law

a = time to start moving

b = speed

D = distance

W = width of target
along axis of motion

60

start
a
1sec

5cm

5cm/s

$$1 + 5 \log\left(1 + \frac{5}{1}\right)$$

$$1 + 5 \cdot 2.58$$

13 sec

T/4 changing

(6)

10 cm/sec

2 sec

10 cm away

1 sec

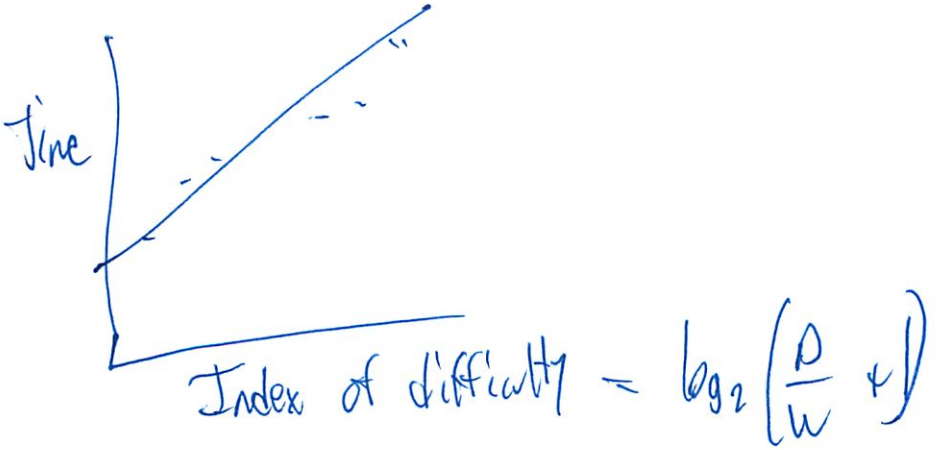
2 cm wide

1 sec

The speed seems weird...

b units = time / bit

No b is found typically...



(7)

So unit for $b = \frac{\text{time}}{\text{bits}}$

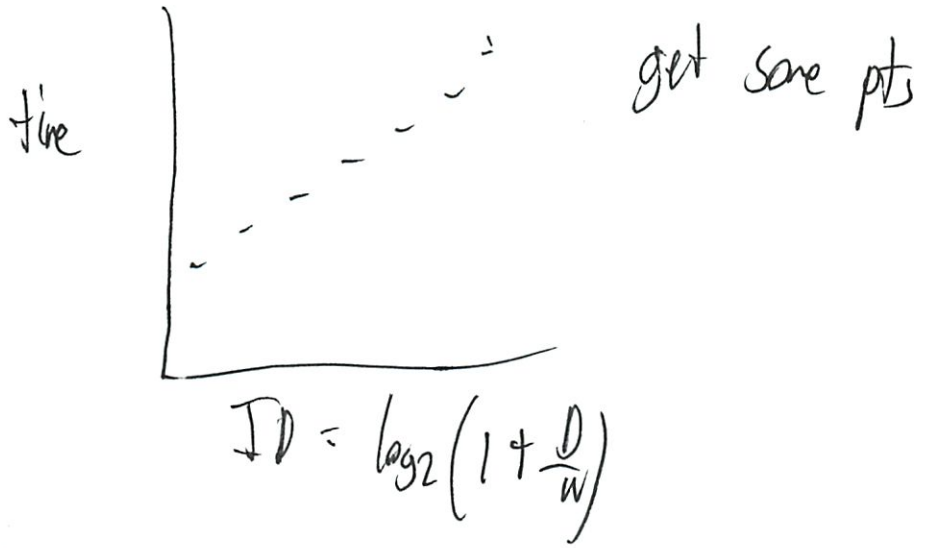
So b is some scaling factor...

$$IP = \frac{1}{b}$$

In 1 class exercise they give $b = 100\,ms$

I still don't see how it all fits together...

You do experiment



I guess this experiment lets you predict new pts which is the goal

8

Ok now what to say?

Tunneling

$$T = a + b\left(\frac{D}{S}\right)$$

?why no log?

Must constantly be smaller than target

length = D

width = S

State why the property is good....

1 bit per sec

$$= \frac{1 \text{ bit}}{\text{sec}}$$

$$1 \Big/ \frac{1 \text{ bit}}{\text{sec}} = 1 \frac{\text{sec}}{\text{bit}} = 1000 \text{ ms/bit}$$

(9)

3 bits per sec

= ,33 sec per bit

= 333 ms per bit

So 100 ms too small

Stylus

7 bits per sec

↳ 142 ms/bit

So what we D,S units

↳ or doesn't matter since will cancel

12 cm long

15 wide

Oh duh $\frac{26}{1}$

(10)

Mase

4

25 yeah close to what I got.

(I think I might have actually figured this out!)

(should have done sibility studies....)

Oh well...

Shannon entropy

$\therefore \log_2(26) \therefore$

4.7

$26 + 26 + 10 + 10$

6.19

(10)

Could add Black Or DD thing to comparison
Would be interesting
kinda late to do so...

---

2 observations:

$\frac{39}{\cancel{2}54}$ possibilities

$\left(\frac{39}{254}\right)^2$ right?

Chance those 39 overlap...
Is that right?

(18)

Ok think about this again

The P(a random len-4 seq matches PIN)

$$= \frac{39}{25^4}$$

So P(a random seq matches 2 PIN codes)

$$= \left(\frac{39}{25^4}\right)^2$$

but we want any seq to match

$$39 \cdot \left(\frac{39}{25^4}\right)^2$$

or $39 \cdot 39 \cdot \left(\frac{39}{25^4}\right)^2$

(Should ask Some one...)

(13)

No prob (a seq matches a Pin is —)
Something else

That is prob(a PIN matches a particular random seq/
(right)

P(2 PINs match a random seq/

$\left(\frac{39}{254}\right)^2$

12/13
4PM

CrossPassword: Novel Password Systems Where Enter Derivation of Password instead of Actual Password

6.858 Final Project

Michael Plasmeier <theplaz>
Jonathan Wang <jwang7>
Miguel Flores <mflores>

Motivation

The problem with many password systems is that users must type their entire, full password each time they log on. This makes the password vulnerable to key logging, shoulder surfing, and interception during transmission.

We explore systems in which the user does not enter their direct password, but a derivation of the password **which changes on each log in**. The user proves that he or she knows the password without subsequently ever providing the password itself.

ING Password Keyboard

A simple example is ING Direct's PIN pad. Under ING's system, the user enters the letters corresponding to their PIN instead of the PIN itself. The mapping between numbers and letters is randomly generated on every log in. This method does not survive an attack where the attacker has access to the mapping, but it does prevent simple keylogging.

Figure 1 ING's Pin Pad. The user enters the letters corresponding to their PIN in the box.

Answering Questions

One could answer questions about the password, instead of inputting the password itself. For example, say a user's password is "tennis ball." The system could prompt "what color is it?" The user would

respond "green." The next time the system could ask "what shape is it?" The user would respond "round." This way the user only transmits their actual object during registration, but never during log in.

Hashing a Response

In an ideal world, the user could prove to the server that it knew the secret by producing a cryptographic hash of the user's secret combined with a server-selected nonce.

$$Hash(Secret_{User}, Nonce_{Server})$$

However, people are not particularly good at being able to calculate cryptographic hashes in their heads, so we need to seek an alternate system.

Inspiration

Original Off the Grid

We were inspired by the "Off the Grid" system from the Gibson Research Corporation.[1] The "Off the Grid" proposal is designed to allow users to use a personal printed paper grid to encipher the domain name of the website they are currently on into a string of psudeo-random characters.

The Off the Grid system works entirely on the user's side. Websites do not need to do anything to support Off the Grid.

To use Off the Grid, the user first generates a grid from a grid-providing website such as https://www.grc.com/offthegrid.htm. This website generates a grid using client-side scripting (ie. JavaScript) to generate the grid on the user's machine. The user then prints the grid onto a sheet of letter paper. At this point the Grid is offline and thus impossible to access by malware. As an alternative, there is at least one application for Android which produces and stores a grid; however, the grid is now accessible to malware on the Android phone which is able to defeat the inter-process sandboxing.

The grid that is generated is a Latin Square. A Latin Square is an $n \times n$ array filled with n different symbols, each occurring exactly once in each row and exactly once in each column.[2] The most famous Latin Square is the popular puzzle game Sudoku. (Note however, that we do not divide up the grid into 9 smaller 3x3 mini-squares in which each symbol must be unique). For example, here is a 11x11 Latin Square with 11 alphabetic characters: Normally, a 26x26 Latin Square is used.

g	e	a	m	o	n	z	k	i	r	c
k	a	n	c	m	z	o	r	g	i	e
n	k	c	z	a	m	r	g	o	e	i
z	o	i	a	n	g	e	c	r	k	m
m	r	z	n	g	a	k	i	e	c	o

[1] https://www.grc.com/offthegrid.htm and associated pages. Is still marked as "Work in Progress;" Retrieved 12/2/2012

[2] http://en.wikipedia.org/wiki/Latin_square

a	g	e	i	z	r	n	o	c	m	k
r	c	g	k	e	i	m	n	z	o	a
e	n	k	g	i	o	c	z	m	a	r
i	m	o	e	r	c	g	a	k	n	k
c	i	r	o	k	e	a	m	n	z	g
o	z	m	r	c	k	i	e	a	g	n

Figure 2 An 11x11 Latin Square; normally 26x26, but reduced in size here to save space.

Once the user has a grid, they use the grid to create or change the password for each website. The Off the Grid specification has a number of variants, but we will use the base variant described on the GRC website. The author recommends that each user adopt slight variations to the rules in order to increase security. To provide a consistent analysis, we assume the user ignores this suggestion.

In the Off the Grid specification, the user traces the name of the website twice to provide additional entropy. In the start of the first phase, the user always starts along the first row of the grid.

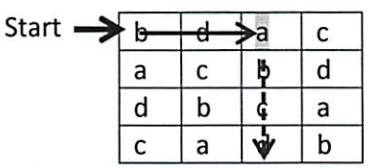


Figure 3

The user then traces out the first 6 characters of the domain name. 6 characters was chosen by the author to provide a 12 character password, which the author chose to balance ease of use with entropy. Again, a user may choose their own scheme. The user alternates between looking horizontally and vertically.

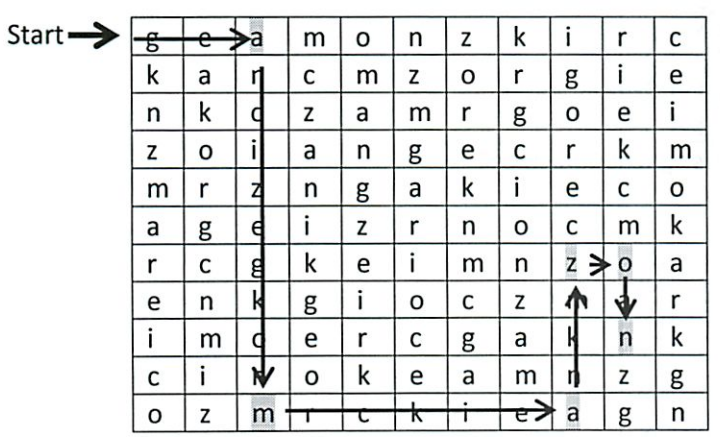
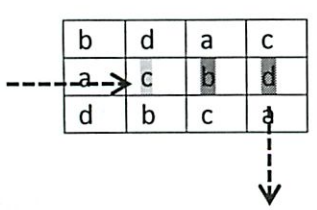


Figure 4

In the second phase, the user starts at the character that they ended with at the end of Phase 1. The user then two more characters from the grid in the same direction of travel. The user then appends those two characters to their password.



c	a	d	b

Figure 5 The user arrives at c traveling to the right. The user appends the next two characters "bd" to their password, and then continues up/down from the last character they read "d".

The user wraps around if their characters go off the grid.

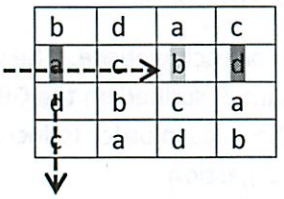


Figure 6 The user arrives at b traveling to the right. The user appends the next two characters to their password, wrapping around if they go off the edge of the grid. Here those characters are "da". The user then continues up/down from the last character "a".

For example here is Phase 2 of our Amazon example.

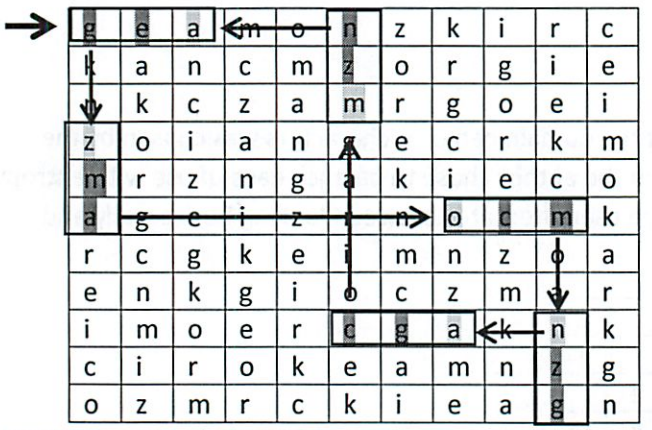


Figure 7 Phase 2 of Off the Grid. The password is "gaznegmacmzg"

Here are Phase 1 and Phase 2.

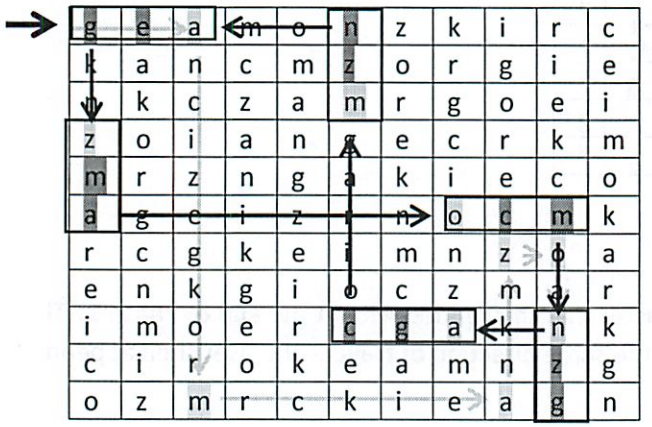


Figure 8 Phase 1 and 2 of Off the Grid.

To log in, the user retraces exactly the same steps as when creating a password. This means the password is exactly the same for each domain. This is an obvious requirement for a system designed to fit within the existing password infrastructure. However, we wanted to explore ideas in which the user does not enter the same password each time.

Description of System

CrossPassword

We wanted to design a system similar to the Off the Grid system, but where the password the user transmits over the network is different each time. With this system, the website presents the user with a grid and the user enters only a deviation of their password.

When the user creates an account, he provides his or her password to the webserver. The user may use characters from the lower case Latin alphabet [a...z]. The password may not have consecutive repeating characters, for example, "aardvark" has the repeating characters "aa" so it would not be allowed. The password is stored on the server such that the plain text can be accessed in order to verify the trace.

When the user logs in, the server randomly generates a 26x26 Latin square with the characters [a...z] called the *Grid*. The server also randomly selects a start row or column called the *start location*. The server transmits this Grid to the user. The Grid and the start location are unique for each log in. The server stores the Grid and start location in temporary state and provides a pointer to this state called the *token* to the user. The user's browser returns the token to the server on each log in attempt.

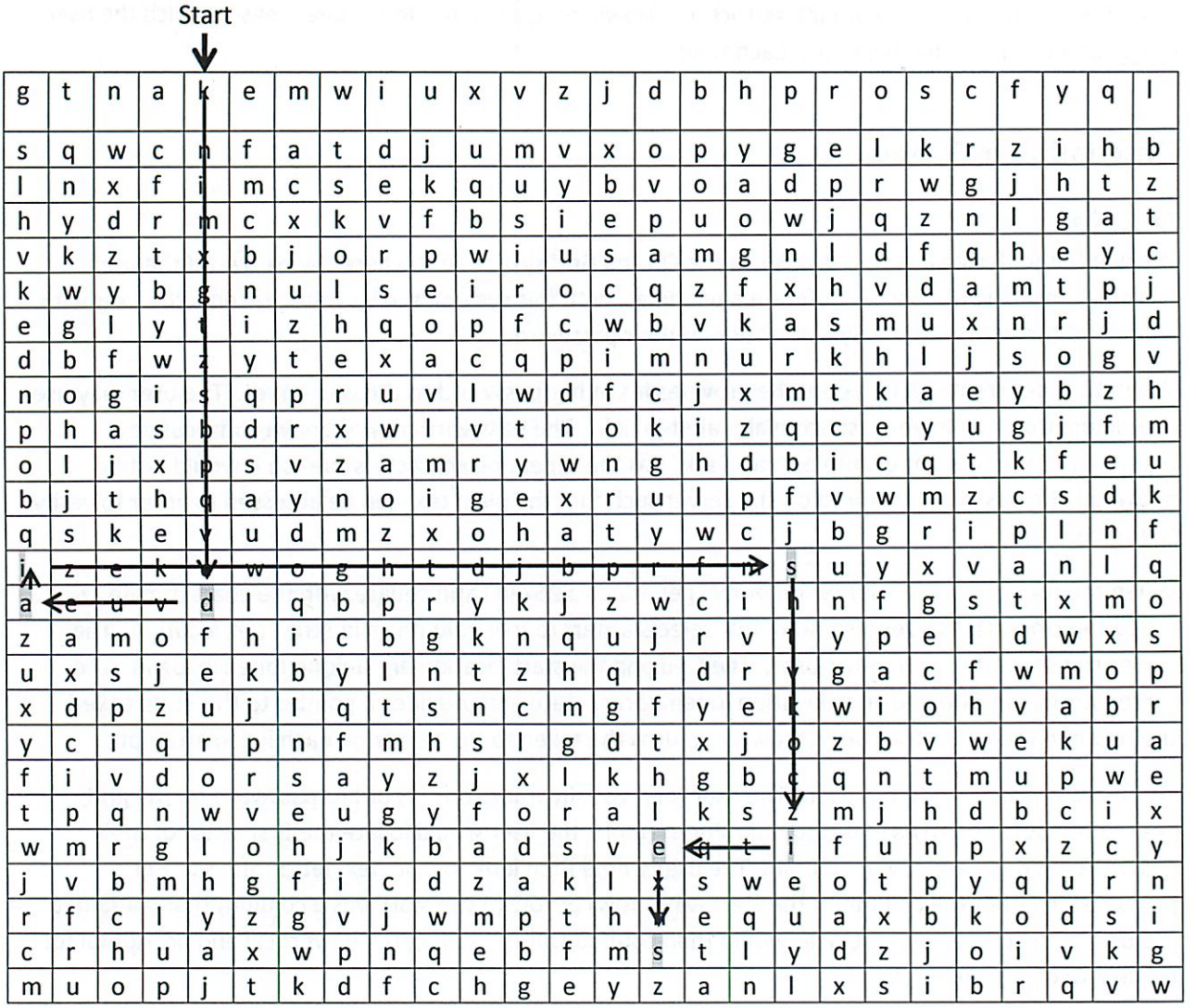
These are transmitted to the user. The user then visually traces out his or her password on the grid, alternating between rows and columns. For example, the user would locate the first letter of their password on the start row or column. The user would then look for the next letter of his or her password in either the column (if the start was a row) or row (if the start was a column) that contained the user's first character. The user would then continue alternating between vertical and horizontal for the length of their password.

The user enters the directions (up, down, left, right) that they follow as they trace out their password. This is called the *trace* of the password. The trace and the token are sent back to the server.

The server verifies that the trace by replaying the trace and making sure the password letters match the provided trace.

The server will only accept 2 traces per token. If a user guesses incorrectly twice, the server will present the user with a new Grid and Start Location. The server will lock the account and the IP address after four incorrect tries until the user completes an email loop.

Example: entering the password Amazon with the 5th column as the start row/column. The grid as well as the start row/column are randomly generated be the server for each log in.



The resulting trace would be: Down, Left, Up, Right, Down, Left, Down.

Figure 9 A trace of the password "daisies"

Modified CrossPassword

We also explored a modified version of this system designed to increase usability. This system uses a 13x13 grid, instead of a 26x26 grid to make it easier for users to visually scan the grid.

In addition, we no longer generate a Latin Square. Instead, we first randomly distribute the user's password in an empty grid. We first randomly select either a row or a column from our 26 choices. We then place the first letter somewhere in that row or column. For this example, say we select the 3rd column to start with. We then place the "a" somewhere in this first column. We then place the second

letter "m" in the row in which we have placed the first letter. We continue this scheme until the password has been placed.

For example:

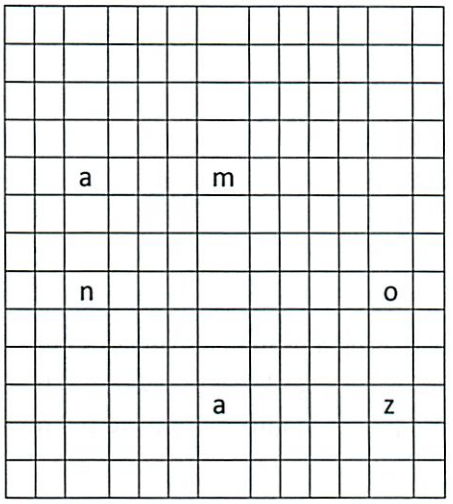


Figure 10

We then randomly fill in the remaining letters on the grid from the set of 26 lower case letters. We make sure each row and column only contains each letter only once by backtracking. For each spot we first start with the entire set [a...z]. We then remove all letters that are currently in the same row and column that we are in. We then randomly select a character from the remaining set.

This is not a Latin Square because we have a 13x13 Grid, but 26 possible characters.

With this system we must prevent an attacker from looking at a grid a certain amount of times. A user can only look at up to 4 random grids before their IP address is locked out. In addition, a particular user account can only have 4 grids shown before that account is locked out as well, in the event the attacker is using a distributed attack.

GrIDsure

As a comparison, we will evaluate our proposal against GrIDsure as described in The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. GrIDsure is also a cognitive authentication scheme in which the user attempts to prove to the server that it knows a secret without actually revealing the secret.

At registration, the user is presented with a 5x5 grid and selects a pattern of 4 cells.

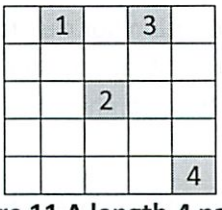


Figure 11 A length-4 pattern selected

At login, the user is again presented with a 5x5 grid, this time with a random digit in each cell. The user then transcribes the digits from his or her length-4 pattern into the nearby box. The digits differ each time because they are chosen randomly each time.

5	3	4	8	9
2	2	7	4	4
3	6	9	6	3
8	4	6	7	0
9	8	1	0	7

Figure 12 The password here would be "3987"

Probabilistic Analysis

We evaluate the security of the three systems described above using a probability analysis. In eavesdropping on a user, an adversary may obtain the user's Square board, the password or trace, or both, the board and the password / trace. Given each of these pieces of information we see how the user may be compromised.

Original Off the Grid

The Off the Grid system offers users a unique password given the domain name. The user simply traces the domain name in the two phases described above.

The first attack may occur if an adversary obtains a user's grid. This is usually held on hand by the user, and may occur if their wallet is stolen.

g	e	a	m	o	n	z	k	i	r	c
k	a	n	c	m	z	o	r	g	i	e
n	k	c	z	a	m	r	g	o	e	i
z	o	i	a	n	g	e	c	r	k	m
m	r	z	n	g	a	k	i	e	c	o
a	g	e	i	z	r	n	o	c	m	k
r	c	g	k	e	i	m	n	z	o	a
e	n	k	g	i	o	c	z	m	a	r
i	m	o	e	r	c	g	a	k	n	k
c	i	r	o	k	e	a	m	n	z	g
o	z	m	r	c	k	i	e	a	g	n

Figure 9 Blank User's Grid

If an attacker is able to retrieve the grid of a user, he obtains no information about the user's password. If the attacker does not know of the Off the Grid system, the grid will offer no information nor clue to the user's password. If however, the attacker does know the Off the Grid protocol and that the user uses it, then all passwords are compromised because the attacker can then follow the two phases described in order to get the password for any site such as Amazon.com.

If an attacker instead obtains the user's password for a single site, only interactions with that website are compromised. In the description described above, the password "gaznegmacmzg" is obtained by following the two phases of the Off the Grid system for Amazon. If the attacker obtains

"gaznegmacmzg," unless he knows which website it belongs to, is useless, otherwise he can use it to log in the user's account for Amazon.

However the attacker obtains no information regarding other passwords for other website domains. In order to do so, the user must guess the grid that created the password. Given the Off the Grid Implementation described, there are at least $\frac{(n!)^{2n}}{n^{n^2}}$ boards for an n-sized board, which leads to at least 9.337×10^{426} boards for $n = 26$. Given this, there is still little information obtained to gaining the password information for any other site.

Lastly, if an attacker retrieves the password and the grid, the user is equally as susceptible to the attacks described in (1). The adversary can now trace out the password in order to determine the website the password belongs to, and if he knows of the Off the Grid system, can retrieve the password for all other websites.

The safety of this implementation relies on how much the user secures the grid. If the grid is stolen, then all of the user's passwords are compromised.

CrossPassword

Our CrossPassword implementation relies on the security of the Latin Square. Since each password must be made up of only lower case letters and no-repeating characters, the password can be searched using a 26×26 Latin Square as specified above, resulting in at least 9.337×10^{426} possible grids.

If the attacker gains access to the board the user sees along with the start location, the attacker never gains any information on the password of the user even after multiple board configurations are given. Each board will always contain all 26 letters, and no information is ever gained about the password.

If the attacker instead gains only access to the user's key-logs, thus obtaining their input, they can never retrieve the password. The password will be impossible to obtain from only getting "up, right, up, left, down" and so on. The only information gained is the length of the password which is 1-to-1 with the trace. Even with multiple traces, the password will be impossible to obtain without the trace. An adversary can then do a brute-force guess on the board because he knows the length of the password. A password presented as Down, Right, Up, Right, Up, Left we know will give us a password of length 6. Additionally, we know that the first element is given through the start-location on the board. We can use the fact that the direction changes each turn, thus Left and Right will always be followed by Up or Down. Thus this gives us a $1 \times 2 \times 2 \times 2 \times 2 \times 2 = 32$ possible combinations which we can then brute force. Thus after finding out the length of a password through the trace, we know there are then 2^{l-1} possibilities given, l, the length of the password.

Brute force is reduced by the frequency of the board change and lockout. So there is a chance of guessing a specific user's password of

$$1 - \left(\frac{31}{32} * \frac{30}{31} * \frac{31}{32} * \frac{30}{31}\right) = 12\%$$

This means an attacker will likely figure out the password of about every ninth user if the attacker is able to use a range of IP addresses, since IP blocking.

Lastly, the most important case is when an attacker gains information to the board, the start-location, and the trace. To begin with,

Brute Force

I would add subheadings

Recover Password

However it is much more difficult for the attacker to backsolve the password back from the grid. Move content from above here

I would at least try to discuss how you know can't be in certain rows, to show that its difficult

Bound on information needed

2 character password example

Is pretty difficult, correct?

What happens if a user picks a dictionary word?

Modified CrossPassword

While our CrossPassword implementation relies on the security of the Latin Square, the modified version is no longer a Latin Square having a reduced size, 13×13, but still using all 26 letters as possibilities.

Here the adversary does not even have to attempt to log in; they can just look at a number of boards

Determine by elimination which character is never missing (and plus frequency analysis as a bonus)

Use past boards; go letter by letter

In 10 refreshes, chance of getting the password is (calculation)

Unlike our previous implementation, the adversary gains information looking at multiple boards over time. One board offers no information about what letters are in the password, but using multiple boards, an adversary may use the boards to determine which letters are in the password. Under some password configurations, one of the letters of the alphabet may not appear in the grid. If we continue onwards with this, we will eliminate all letters that are not in the password. It is then a matter of determining the length of the password and see what are plausible passwords given the letters.

Similar to our Original CrossPassword implementation, the adversary gains no information when multiple traces are presented other than the length of the password. They can then brute force the password knowing there are 2^{l-1} possible combinations.

Lastly, there is the possibility of having all information of the board, the start-location, and the trace.

Why only 4 random grids?

Needs ~~1-2 traces~~ and 5-10 grids, correct? No trace needed, I think

GrIDsure

There are 25^4possible combinations of length-4 patterns.

However, there are 10^4 possible inputs to return.

We cannot exactly map one "trace" from the grid to the squares that the user selected because there are 25 grid locations but only 10 digits. Thus each digit will be in the grid an average of 2.5 times.

How many Grids and answers do we need?

Since each item is on the grid an average of 2.5 times there are 2.5^4 possible underlying grids for each given trace. That means that with only 1 observation, an attacker as a 1 in 39 chance of guessing the original box pattern correctly.

If the attacker has two observations, then there is a very small chance (how small) that more than two of the 39 possible grid patterns overlap. There will be our 1 target overlap, but a very, very small chance of accidential overlap.

Can you check this?

P(a PIN matches a particular random seq) = $\frac{39}{25^4}$

P(2 PIN matches a particular random seq) = $(\frac{39}{25^4})^2$=$9*10^{-9}$ randomly

(how do the math?)

Other Factors

We evaluate each system according to the criteria set out in The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes.[3]

Improvements to Criteria

Resilient-to-Physical Observations Category

We think that the **Resilient-to-Physical Observations** category should be split in two: **casual observation** and **video observation**. Casual observation is if an attacker is just able to watch the user enter their password once. This is feasible for short passwords and/or if the user types slow. An attacker can see which keys are hit on the keyboard. This is especially true if the user types slowly, has a short, and/or easily remember-able password.

However, the attacker seeing the user trace out the password on the grid once would have trouble remembering the entire grid, preventing the total loss of the password scheme. For that specific

[3] Bonneau, Joseph, Cormac Herley, Paul C. van Oorschot, Frank Stajanoy. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. University of Cambridge; Microsoft Research; Carleton University; University of Cambridge. Proc. IEEE Symp. on Security and Privacy 2012 ("Oakland 2012"). http://css.csail.mit.edu/6.858/2012/readings/passwords.pdf

domain name, many attackers would have trouble remembering the sequence of 12 random characters, providing some additional security.

Video observation is defined as the attacker having the full ability to carefully watch and study users' movements because the attacker is able to pause and replay the user's log in actions.

Resilient-to-Throttled-Guessing

To better demonstrate the differences between our protocols, we assign **Resilient-to-Throttled-Guessing** if there are more than 10 possible choices all of equal weight. This is much small than the original paper requires. The paper considers $\frac{1}{10^4}$ choices to be NOT **Resilient-to-Throttled-Guessing**.

Inherently-Discoverable

Must the user seek out the new password system? Or does the server require that the user use it? Often new schemes that fit within the structure of existing passwords remain undiscoverable to the user. We want to highlight schemes where the website helps the user discover them. A scheme gets a YES here if the server is required to notify and teach the user of the new scheme.

Resilient-to-SSL-Proxy-Man-in-the-Middle

Assume that there is someone who is listening in on the wire who can decrypt SSL, for example, a corporate SSL proxy. Does this person have enough information to log in? YES, if they can do so after observing 1 log in. QUASI, if they must observe several log ins in order to have this power. We assume that the initial registration process is outside this scheme.

Allows -User-to-Choose-Any-Password

Does the system allow the user to choose any password (as defined by the usual set of characters allowed in a password)? Or does the system limit the user's set of password to a certain length or set of characters? Or use a totally different memory scheme? Users might use the same password on multiple sites or have an external scheme to generate a password. One could argue that preventing a user from using the same password on each site is a good thing, but a password scheme should not do so by limiting the choice that a user has in selecting a password.

Denial-of-Serviceable

An active attacker can cause a denial-of-service attack by submitting a sufficient quantity of incorrect passwords such that the system locks the user out. Lockouts can add additional security by preventing more than a handful of guesses by the attacker. However, they can considerably impede usability as they can require a user to either wait or to seek out help from a system administrator. If these are tied to a user account, an attacker can deliberately use up these guesses to mount a Denial-of-Service attack on the user. If this is possible with a few incorrect submissions from any IP address, we assign a YES here.

Original Off the Grid

Usability benefits

1. **Memorywise-Effortless YES** There are no secrets to be remembered in the base case. The description mentions a more advanced case, where the user could start at a different location, but we are assuming the base case where the user automatically selects the same location.

2. **Scalable-for-Users YES** The user only needs one grid for all of their sites.
3. **Nothing-to-Carry NO** User must carry 1 sheet of paper
4. **Physically-Effortless NO** The user must trace out their password on paper
5. **Easy-to-Learn NO** Using the same rubric as the paper does, the scheme is quite complicated
6. **Efficient-to-Use NO** The scheme requires a fair amount of effort for each authentication.
7. **Infrequent-Errors NO** Tracing out the password on the grid twice is easy to mess up.
8. **Easy-Recovery-from-Loss KINDA** If the user lost their Grid, they must have another copy of their Grid, or the key used to generate that Grid. A user can always reset their passwords on each site. The paper rates generic passwords as **Easy-Recovery-from-Loss YES**.
9. **Inherently-Discoverable NO** A user must learn about this scheme by visiting the GRC website.
10. **Allows -User-to-Choose-Any-Password NO** The password is based off of the domain name of the site.

Deployability benefit

1. **Accessible NO** There could conceivably be a braille-based grid, but not at this moment. In addition, someone with poor motor control will find this scheme very difficult.
2. **Negligible-Cost-per-User YES** The user is required to print one sheet of paper which costs < 05 cents.
3. **Server-Compatibility YES** One of the primary benefits of this scheme is that it is compatible with existing servers which use passwords
4. **Browser-Compatibility YES** No special browser is needed
5. **Mature KINDA** The scheme has been published for some length of time; at least one Android app exists with support.
6. **Non-Proprietary YES** The scheme is published fully.

Security benefits

1. **Resilient-to-Physical Observations-Casual KINDA** The attacker would have to remember 12 random characters in order to observe the user's password for that site.
   With just a casual observation there is no way the attacker can memorize the entire Grid.
2. **Resilient-to-Physical Observations-Video NO** If the attacker can take a picture of the Grid, for example, a video camera over the shoulder, then the attacker would have access to all of the users' passwords assuming the user is using the standard Of the Grid scheme.
3. **Resilient-to-Targeted-Impersonation YES** Personal knowledge cannot help for the Off the Grid scheme. However, the normal password recovery mechanisms of the website remain, which are generally very vulnerable to Targeted Impersonation.
4. **Resilient-to-Throttled-Guessing YES** The user's password is 12 random alphanumeric characters. This means there are 26^{12} possible passwords.
5. **Resilient-to-Unthrottled-Guessing YES** There are 26^{12} possible passwords.
6. **Resilient-to-Internal-Observation NO** Off the Grid reduces to a normal 12 character password unique for each domain. This password is the same for each log in.
7. **Resilient-to-SSL-Proxy-Man-in-the-Middle NO** The password is the same for each log in; it must be protected with some additional protection (such as SSL) in transit.
8. **Resilient-to-Leaks-from-Other-Verifiers YES** Ideally the server should be hashing the password. Regardless, each domain has a unique password so leaking one password does not give one feasible information about another domains' password.

9. **Resilient-to-Phishing NO** If the attacker is able to spoof the domain name of the site, then the user will follow the same trace on the grid, providing the attacker their password.
10. **Resilient-to-Theft NO!** If the attacker gets your grid, it's game over, assuming you are sticking to the base Off the Grid algorithm. The author suggests that you make small personal tweaks to the algorithm in order to add resilience to theft.
11. **No-Trusted-Third-Party YES** The third party provides the code to generate the grid. However, that code runs in JavaScript on your local computer, allowing you to verify that the code is actually generating a unique grid and is not sending a copy to the third party. One could also write ones' own implementation of the Grid generation scheme to be sure.
12. **Requires-Explicit-Consent YES** The user must trace their password on the grid and then enter it onto the computer.
13. **Unlinkable YES** Since each user's Grid is so different, there is no feasible way to link users using the same scheme.
14. **Denial-of-Service-able NO** This is the same as normal passwords. Under a normal password system, services generally do not add a lockout provision.

CrossPassword

Goal: prevent from seeing over wire

Note all are for the actual log in experience. This analysis does not consider creating a password; the process of which is similar to traditional password schemes.

Usability benefits

1. **Memorywise-Effortless NO** The user must remember a password to use CrossPassword. Ideally, that password should be different between sites. Since we only allow lowercase alphabetic characters without repeating letters, we may prevent users from using the same password on a site running CrossPassword than the user uses on all of their sites.
2. **Scalable-for-Users NO** Ideally the user has a different password for each site
3. **Nothing-to-Carry YES** There is nothing to carry
4. **Physically-Effortless NO** The user must trace out their password on-screen
5. **Easy-to-Learn NO** Using the same rubric as the paper, the scheme is quite complicated
6. **Efficient-to-Use NO** The scheme requires a fair amount of effort for each authentication.
7. **Infrequent-Errors NO** Tracing out the password on screen is easy to mess up
8. **Easy-Recovery-from-Loss YES** CrossPassword falls back on the same recovery mechanisms as traditional password sites, which is rated YES in the paper.
9. **Inherently-Discoverable YES** A user will discover the CrossPassword scheme when attempting to create an account on a server that uses CrossPassword
10. **Allows -User-to-Choose-Any-Password NO** The user can only choose a password using the letters [a...z] and the user cannot repeat the same characters twice, as in "aardvark."

Deployability benefit

1. **Accessible NO** A screen reader would be tedious to use. In addition, someone with poor motor control will find this scheme very difficult.
2. **Negligible-Cost-per-User YES** There is no cost.
3. **Server-Compatibility NO** The server must be provisioned with a new authentication library.

4. **Browser-Compatibility YES** No special browser is needed
5. **Mature NO** We are proposing it here
6. **Non-Proprietary YES** The scheme is published fully.

Security benefits

1. **Resilient-to-Physical Observations-Casual POSSIBLY** If the attacker could see the screen and the keyboard they could not uncover the user's password, unless the user traces the password with their finger.
2. **Resilient-to-Physical Observations-Video POSSIBLY** Even with being able to study the user as they enter their password, the attacker would not be able to recover a user's password, unless the user traces the password with their finger. This is one of the major design goals of this system.
3. **Resilient-to-Targeted-Impersonation YES** Personal knowledge cannot help for the Off the Grid scheme. However, the normal password recovery mechanisms of the website remain, which are generally very vulnerable to Targeted Impersonation.
4. **Resilient-to-Throttled-Guessing YES** An attacker can only submit two tracers per grid/start location. After two tries, the server will issue a new grid. The user then gets two more tries at a trace submission before the account is locked until an email loop is performed.
5. **Resilient-to-Unthrottled-Guessing NO** Due to the very small number of possible responses (for example, 2^5=32 for a 6 character password, there are very few bits of entropy so the system falls fast.
6. **Resilient-to-Internal-Observation YES** This is the major design goal of this system. An attacker needs many observations of the *grid*, *Start Location*, and *trace* in order to crack the password. This is sharply reduced if the user picks a dictionary word, however.
7. **Resilient-to-SSL-Proxy-Man-in-the-Middle YES** This is the same as Internal-Observations. If a listener on the wire who was able to remove the SSL encryption, then they would need several observation in order to recover the password.
8. **Resilient-to-Leaks-from-Other-Verifiers NO** The password is stored in plain text on the server in order for the server to verify the password. This is not good practice.
9. **Resilient-to-Phishing YES** An attacker with just one trace could not submit that trace to another server, because the grid is randomized each time. At attacker could mount a man-in-the-middle attack and proxy the grid, but the rubric in the paper does not penalize for this.
10. **Resilient-to-Theft YES** There is nothing to steal
11. **No-Trusted-Third-Party YES** There are no 3rd parties involved
12. **Requires-Explicit-Consent YES** The user must trace their password on the computer and enter the trace.
13. **Unlinkable YES** Like passwords, this scheme is unlinkable.
14. **Denial-of-Service-able YES** An attacker can lock out an account by trying an incorrect password 4 times.

Modified CrossPassword

The modified CrossPassword is more **Efficient-to-Use** and has less errors (**Infrequent-Errors**), however at the cost of a slightly decreased **Resilient-to-Physical Observations-Casual** and **Resilient-to-Physical Observations-Video** if a user traces the grid because of the smaller grid. The degree is reduced, but the broad scores remain the same.

However, **Resilient-to-Internal Observation** and **Resilient-to-SSL-Proxy-Man-in-the-Middle** take big hits as an attacker can discover a user's password (or at least have a very high chance of finding it) using 5-10 copies of the random grid and start location. They don't need any copies of the trace, though having at least 1 would help. This makes the scheme vastly weaker. **Resilient-to-Throttled-Guessing**, **Resilient-to-Unthrottled-Guessing** switch to no because the attacker has that high chance of recovering the password and might only need to make 1-2 guesses. In fact, we don't even have a category for how bad this is: **Crackable-From-Reloading-Log-In-Page**?

Because we now need to protect from the attacker seeing the grid multiple times, modified CrossPassword is even more **Denial-of-Service-able.**

GrIDsure

GrIDsure is evaluated in The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. Here we evaluate the new metrics we have introduced and make additional comments about some metrics.

It is important to note that the authors rated it as not **Resilient-to-Throttled-Guessing** or **Resilient-to-Unthrottled-Guessing** because the space of possible is so small (10^4). In this paper, we rated our other schemes as **Resilient-to-Throttled-Guessing** if the attacker has less than 10 possible choices, so we would rate this as **Resilient-to-Throttled-Guessing** if it has a rate limiter/lockout.

Because the server must tell the user about the scheme, GrIDsure is **Inherently-Discoverable**. However, it requires users to remember a sequence of 4 unmarked boxes in a 5x5 grid. Thus it is clearly not **Allows -User-to-Choose-Any-Password.**

When the user does not place their finger to the screen, it is **Resilient-to-Physical Observations-Casual.**

Because of the small number of possibilities it is not **Resilient-to-Internal-Observation**, **Resilient-to-SSL-Proxy-Man-in-the-Middle**, or **Resilient-to-Physical Observations-Video**. With two observations, it is pretty much game over, as the attacker is able to discover the original sequence of boxes.

Comparison Table

	Of the Grid	CrossPass word	Modified CrossPass word	GrIDsure
Memorywise-effortless	Yes	No	No	No
Scalable-for-users	Yes	No	No	No
Nothing-to-carry	No	Yes	Yes	Yes
Physically-effortless	No	No	No	No
Easy-to-Learn	No	No	No	Yes
Efficient-to-Use	No	No	No (More)	Quasi
Infrequent-Errors	No	No	No (More)	Quasi
Easy-Recovery-from-Loss	Kinda	Yes	Yes	Yes
Inherently-	No	Yes	Yes	Yes

Discoverable				
Allows -User-to-Choose-Any-Password	No	No	No	No
Accessible	No	No	No	No
Negligible-Cost-per-User	Yes	Yes	Yes	Yes
Server-Compatibility	Yes	No	No	No
Browser-Compatibility	Yes	Yes	Yes	Yes
Mature	Kinda	No	No	No
Non-Proprietary	Yes	Yes	Yes	No
Resilient-to-Physical Observations-Casual	Kinda	Possibly	Possibly (Less)	Yes
Resilient-to-Physical Observations-Video	No	Possibly	Possibly (Less)	No
Resilient-to-Targeted-Impersonation	Yes	Yes	Yes	No
Resilient-to-Throttled-Guessing	Yes	Yes	No!	Yes
Resilient-to-Unthrottled-Guessing	Yes	No!	No (Less)	No
Resilient-to-Internal-Observation	No	Yes	No!	No
Resilient-to-SSL-Proxy-Man-in-the-Middle	No	Yes	No!	No
Resilient-to-Leaks-from-Other-Verifiers	Yes	No!	No!	No
Resilient-to-Phishing	No	Yes	Yes	No
Resilient-to-Theft	No!	Yes	Yes	Yes
No-Trusted-Third-Party	Yes	Yes	Yes	Yes
Requires-Explicit-Consent	Yes	Yes	Yes	Yes
Unlinkable	Yes	Yes	Yes	Yes
Denial-of-Service-able	No	Yes	Yes (More)	No

Table 1: A Comparison of Off the Grid, CrossPassword, Modified CrossPassword, and GrIDsure.

Usability

We will now explore what the field of usability tells us about our password schemes. [4] The three core tenants of usability are: learnability, efficiency, and safety.

At the core, the simpler a system is, the more it will be used. Security is often a tradeoff between usability and security. A successful scheme should add security, without impacting usability too much.

[4] Material from MIT's 6.813 User Interface classes by Prof. Rob Miller Spring 2012.

Learnability

Discoverability

In order for a system to start being used, it must be discoverable.

CrossPassword is more discoverable than Off the Grid because the website you are creating an account with can let you know that the website uses CrossPassword. It is inherently discoverable. Off the Grid requires that you hear about the system in some way. Websites can still advise you of the presence of Off the Grid, but the Off the Grid system, as currently designed and designated, is not inherently discoverable.

Training

It's important that a user know how to use a particular system.

CrossPassword can be taught to users when they pick their password for the site. For example, sites could show users a video of how to use CrossPassword. Sites could also provide an interactive training tool using CrossPassword that uses JavaScript and HTML 5 to show the user how to trace their actual password. (Using the actual password would reveal the user's password to a shoulder surfing attacker, but this may be appropriate for a secure room. The password would be stored in the DOM during registration, but this happens with a normal registration system as well)

Mental Model

When users interact with a system, they form a mental model of how that system operates "behind the scenes."

We believe that once CrossPassword is explained to a user, it is easy for that user to form a mental model of the system. The server asks you to solve a puzzle and you solve it. In addition, the rationale behind the system is also clear; it is clear that this prevents you from sending your password over the wire for subsequent log ins. Users should be able to understand how the system works. Each log in is consistent with the rules of the system and ones' mental model of the system.

Efficiency

Each log in should not take a long time. This is because user's time is valuable. In addition, users will be more likely to keep using the system if it is fast.

Whereas Off the Grid requires users to trace the grid twice, CrossPassword only requires a user to trace the grid once.

Off the Grid also requires one to enter two characters for each letter in the domain name during its Phase 2. CrossPassword is more natural to use than Off the Grid because one can trace the system on the screen as one enters the keyboard traces. We feel that expert users of CrossPassword could use the arrow keys without taking their eyes off the screen. This could make password entry quite fast.

However, both CrossPassword and Off the Grid are slower than traditional password schemes, or even password managers, such as LastPass. Users may not want to adopt a system that is slower than what they already have.

Chunking

Research has shown that people can remember 7 ±2 pieces of information at once.[5] A piece of information could be one letter. When letters are combined into an English word, that word is now one piece of information. To reiterate: a collection of 7 random letters are 7 pieces of information. However, a word comprised of 7 letters is only 1 piece of information.

We can use this to evaluate whither a causal visual observer (shoulder surfer) could observe a password off the screen. With Off the Grid, it would be difficult for an attacker to remember 12 characters using just their short term memory. This is why we rated it as KINDA for Resilient-to-Physical Observations-Casual.

Fitts's Law

Fitts's Law is an estimate of the time it takes someone to point to an object or steer among objects.[6] The rule as formulated by Scott MacKenzie is as follows:[7]

$$T = a + b \log_2(1 + \frac{D}{W})$$

where:

- T is the average time taken to complete the movement
- a represents reaction time to start moving
- b stands for the speed of movement
- D is the distance from the starting point to the center of the target.
- W is the width of the target measured along the axis of motion. W can also be thought of as the allowed error tolerance in the final position, since the final point of the motion must fall within $\pm W/2$ of the target's center.

We can use a more specific form to study steering tasks, the time to move your hand through a tunnel of length D and width S:

$$T = a + b (\frac{D}{S})$$

The index of difficulty is now linear.

We can use this to measure the amount of time it takes someone to trace through the grid, if they trace the grid with their finger or mouse. Ideally the user should not do that to maintain **Resilient-to-Physical Observations-Casual** and **Resilient-to-Physical Observations-Video**.

---

[5] De Groot, A. D., *Thought and choice in chess*, 1965

[6] Paul M. Fitts (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, volume 47, number 6, June 1954, pp. 381–391.

[7] I. Scott MacKenzie and William A. S. Buxton (1992). Extending Fitts' law to two-dimensional tasks. Proceedings of ACM CHI 1992 Conference on Human Factors in Computing Systems, pp. 219–226. http://doi.acm.org/10.1145/142750.142794

Assume a = 0 and b=200ms/bit for a mouse, using the upper limit of the empirical study.[8] Assume the user must travel 26 cm to reach a 1 cm square block. If a user had to steer within a row, this leads to an approximate time to trace of

$$0 + .2(26) = 5.2 \text{ seconds for traveling a row or column.}$$

This is the worst case possibility: the user (worst case empirical user) is using a mouse to travel the entire length of a row/column and they cannot leave the row/column with their mouse at all. This is

Improving Usability

We can do things to improve usability. For example, we can shade every other row or column, alternating between row and column on every user input.

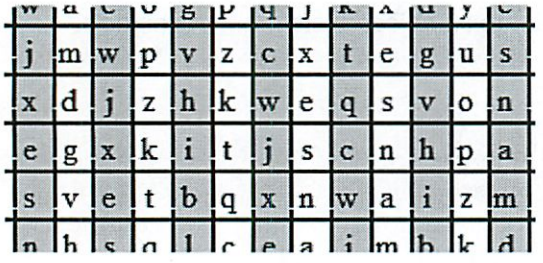


Figure 13 Every other column is shaded

This gives us two benefits. First, the user can now easily see if they should move horizontally or vertically next. Second, it is easier for the user to keep their eye in the same column/row as they scan the grid vertically/horizontally for their next letter. This should decrease mistakes as well as decrease the time it takes people to solve the grid.

Auto-Solver

It is possible to build a browser-based auto-solver for CrossPassword grids. This software would know the user's password and use that to automatically solve grid challenges. This would break **Resilient-to-Internal Observation** because the user's system would now need the password stored. However the system would still meet **Resilient-to-SSL-Proxy-Man-in-the-Middle** . It would do a great deal for usability, flipping **Physically-Effortless**, **Easy-to-Learn**, **Efficient-to-Use**, and **Infrequent-Errors** all to yes. In addition **Accessibility** would greatly improve.

Code (Jwang)

Anything we want to write here?

How do we generate a Latin Square?

Limitations of Current Code

We have not implemented any sort of lockout system in our sample code.

---

[8] Soukoreff, R. William, and I. Scott MacKenzie. Towards a standard for pointing device evaluation, perspectives on 27 years of Fitts' law research in HCI. York University. Department of Computer Science and Engineering. November 4, 2004. http://www.yorku.ca/mack/ijhcs2004.pdf

Conclusion

CrossPassword is not recommended as a password system. Modified CrossPassword turned out to be even weaker than we first imagined.

We tried to build a zero-knowledge interactive proof. A zero-knowledge interactive proof is one in which the prover needs to show that they know the solution to the verifier. The prover in this case is the user, and the verifier is the server. The verifier asks questions to the prover, who responds with an answer. If the prover does in fact know the answer, he or she will always answer the verifier's question correctly. If however, the prover does not actually know the answer, the prover may still get the question correct. However, over many questions the prover is likely to guess incorrectly at some point. Thus after enough guesses it is very unlikely that the prover is faking it. However, we only pose one question on each log in, which is not sufficient for a zero-knowledge interactive proof. Even with a super-aggressive lock out, CrossPassword still has false negative rates are well above the standards for cryptographic algorithms.

We controlled for the wrong thing. The password had a lot of information. However, the trace which we return to the server has very little information. For example, say we take a password and XOR the characters together to get 1 bit which is either yes or no. We transmit very little meaningful information of the password, but that very fact makes it easy for the attacker to guess!

The **Shannon entropy** of CrossPassword is $(l - 1)$ where l is the number of characters in the password. For example, a 6 character password has 5 bits of entropy. This makes it easy to brute force. In comparison, a single letter a-z has 26 possibilities or $log_2(26)$ = 4.7 bits *per letter*. Thus our 6 character password is almost the equivalent of a password of a single letter! If we allow upper and lower case, digits, and 10 special characters, we have $log_2(72)$ = 6.2 bits of entropy, which is more than we currently have!

GrIDsure also has a reduction of entropy from 25^4 choices to 10^4 choices. However, 10^4 represents $log_2(10^4)$ = 13.3 bits of entropy, which is a good deal more than CrossPassword. Remember each additional bit doubles the number of possible passwords, and thus doubles the brute-force password search time.

However, GrIDsure is even unable to fulfill its design goal if an attacker has even two complete observations of grids and the corresponding PIN code.

The Modified Scheme ends up being even worse because we reveal information about the password to the user. This ends up being disastrous because an attacker only needs to receive 5-10 copies of the grid from the server, and no data from the user actually entering their password.

On top of all this, our scheme is slower to enter than a traditional password, especially when used with a password manager.

This shows the inherent complexity in producing password schemes. There are many different objectives to try to achieve at once. Trading off some objectives produces different outcomes in security. Objectives cannot be traded off one-for-one, since the factors are not evenly weighted. There are many different possible attacks on a password scheme. It is difficult to keep all of the possible attacks in mind as one designs a particular scheme. Although this scheme was weak, it was interesting to evaluate exactly why it was weak and to think of possible attacks against the scheme.

12/14
12:30p

My comments on Miguels section

At login, the user is again presented with a 5x5 grid, this time with a random digit in each cell. The user then transcribes the digits from his or her length-4 pattern into the nearby box. The digits differ each time because they are chosen randomly each time.

5	3	4	8	9
2	2	7	4	4
3	6	9	6	3
8	4	6	7	0
9	8	1	0	7

Figure 12 The password here would be "3987"

Analysis of Attacks

We evaluate the security of the three systems described above using a probability analysis. In eavesdropping on a user, an adversary may obtain the user's Square board, the password or trace, or both, the board and the password / trace. Given each of these pieces of information we see how the user may be compromised.

Original Off the Grid

The Off the Grid system offers users a unique password given the domain name. The user simply traces the domain name in the two phases described above.

The first attack may occur if an adversary obtains a user's grid. This is usually held on hand by the user, and may occur if their wallet is stolen.

g	e	a	m	o	n	z	k	i	r	c
k	a	n	c	m	z	o	r	g	i	e
n	k	c	z	a	m	r	g	o	e	i
z	o	i	a	n	g	e	c	r	k	m
m	r	z	n	g	a	k	i	e	c	o
a	g	e	i	z	r	n	o	c	m	k
r	c	g	k	e	i	m	n	z	o	a
e	n	k	g	i	o	c	z	m	a	r
i	m	o	e	r	c	g	a	k	n	k
c	i	r	o	k	e	a	m	n	z	g
o	z	m	r	c	k	i	e	a	g	n

Figure 13 Blank User's Grid

Intercepted Grid

If an attacker is able to retrieve the grid of a user, he obtains no information about any of the user's passwords. If the attacker does not know of the Off the Grid system, the grid will offer no information nor clue to the user's password. If however, the attacker does know the Off the Grid protocol and that the user uses it, then all passwords are compromised because the attacker can then follow the two phases described in order to get the password for any site such as Amazon.com.

Comment [MEP1]: I think we assume the attacker knows the protocol

Observing a Single Site

If an attacker instead obtains the user's password for a single site, only interactions with that website are compromised. In the description described above, the password "gaznegmacmzg" is obtained by following the two phases of the Off the Grid system for Amazon. If the attacker obtains "gaznegmacmzg," unless he knows which website it belongs to, it is useless, otherwise he can use it to log in the user's account for Amazon.

However the attacker obtains no information regarding other passwords for other website domains. In order to do so, the user must guess the grid that created the password. Given the Off the Grid implementation described, there are at least $\frac{(n!)^{2n}}{n^{n^2}}$ boards for an n-sized board, which leads to at least 9.337×10^{426} boards for $n = 26$. Given this, there is still little information obtained to gaining the password information for any other site.

Lastly, if an attacker retrieves the password and the grid, the user is equally as susceptible to the attacks described if he had no grid at all. The adversary can now trace out the password in order to determine the website the password belongs to, and if he knows of the Off the Grid system, can retrieve the password for all other websites.

The safety of this implementation relies on how much the user secures the grid. If the grid is stolen, then all of the user's passwords are compromised.

CrossPassword

Our CrossPassword implementation relies on the security of the Latin Square. Since each password must be made up of only lower case letters and no-repeating characters, the password can be searched using a 26×26 Latin Square as specified above, resulting in at least 9.337×10^{426} possible grids.

Intercepted Board

If the attacker gains access to the board the user sees along with the start location, the attacker never gains any information on the password of the user even after multiple board configurations are given. Each board will always contain all 26 letters, and no information is ever gained about the password.

Recovering User Password from Trace

If the attacker instead gains only access to the user's key-logs, thus obtaining their input, they can never retrieve the password. The password will be impossible to obtain from only getting "up, right, up, left, down" and so on. The only information gained is the length of the password which is 1-to-1 with the trace. Even with multiple traces, the password will be impossible to obtain without the accompanying board.

Brute Forcing Traces

An adversary can do a brute-force guess on the board because he knows the length of the password. If a trace is intercepted as Down, Right, Up, Right, Up, Left, we now know the length of the password, 6. Additionally, we know that the first element is given through the start-location on the board. We can use the fact that the direction changes each turn, thus Left and Right will always be followed by Up or Down. Thus this gives us a $1 \times 2 \times 2 \times 2 \times 2 \times 2 = 32$ possible combinations which we can then brute force. After finding out the length of a password through the trace, we know there are then 2^{l-1} possibilities given, l, the length of the password.

Comment [MEP2]: I think we can assume the attacker knows what site it belongs to. The distinction here is that they gain very little info about other sites. Can we articulate this more clearly?

Comment [MEP3]: Can we articulate this more clearly? Ie. Give some explanation of what the attacker would have to do and why that is impossible?

Comment [MEP4]: I wouldn't say that it's like no grid at all; I would say the user is totally hosed. The system absolutely relies on keeping the grid secure. Again we can assume the attacker knows the system. We could add that if the user adopts a random suggested modification, then they would be somewhat better off. Ie random start location, there are 104 possibilities, which is better than 1, but not very good.

Comment [MEP5]: This seems duplicate with the section above

Comment [MEP6]: It is very difficult

Comment [MEP7]: How does this relate to below paragraph

Because this scheme is highly susceptible to brute force attacks with a $\frac{1}{2^{l-1}}$ probability of guessing correctly, we can reduce the effectiveness of brute force attacks by the frequency of the board change and lockout. If we lockout after every 4 attempts, then there is a chance of guessing a specific user's password of

$$1 - \left(\frac{31}{32} * \frac{30}{31} * \frac{31}{32} * \frac{30}{31}\right) = 12\%$$

before being locked out given a trace of length 6. This means an attacker will likely figure out the password of about every ninth user if the attacker is able to use a range of IP addresses, due to IP blocking.

Intercepted Board and Trace: Password Recovery?

Lastly, the most important case is when an attacker gains information to the board, the start-location, and the trace. Even when the trace is intercepted along with its board, it is extremely difficult to recover the password as can seen in the example below where the password is "daisies."

Comment [MEP8]: I would not agree that it is the most important case

Start ↓

g	t	n	a	k	e	m	w	i	u	x	v	z	j	d	b	h	p	r	o	s	c	f	y	q	l
s	q	w	c	n	f	a	t	d	j	u	m	v	x	o	p	y	g	e	l	k	r	z	i	h	b
l	n	x	f	l	m	c	s	e	k	q	u	y	b	v	o	a	d	p	r	w	g	j	h	t	z
h	y	d	r	m	c	x	k	v	f	b	s	i	e	p	u	o	w	j	q	z	n	l	g	a	t
v	k	z	t	x	b	j	o	r	p	w	i	u	s	a	m	g	n	l	d	f	q	h	e	y	c
k	w	y	b	g	n	u	l	s	e	i	r	o	c	q	z	f	x	h	v	d	a	m	t	p	j
e	g	l	y	t	i	z	h	q	o	p	f	c	w	b	v	k	a	s	m	u	x	n	r	j	d
d	b	f	w	z	y	t	e	x	a	c	q	p	i	m	n	u	r	k	H	l	j	s	o	g	v
n	o	g	i	s	q	p	r	u	v	l	w	d	f	c	j	x	m	t	K	a	e	y	b	z	h
p	h	a	s	b	d	r	x	w	l	v	t	n	o	k	i	z	q	c	E	y	u	g	j	f	m
o	l	j	x	p	s	v	z	a	m	r	y	w	n	g	h	d	b	i	C	q	t	k	f	e	u
b	j	t	h	q	a	y	n	o	i	g	e	x	r	u	l	p	f	v	w	m	z	c	s	d	k
q	s	k	e	v	u	d	m	z	x	o	h	a	t	y	w	c	j	b	g	r	i	p	l	n	f
i	z	e	k	c	w	o	g	h	t	d	j	b	p	r	f	m	s	u	y	x	v	a	n	l	q
a	e	u	v	d	l	q	b	p	r	y	k	j	z	w	c	i	h	n	f	g	s	t	x	m	o
z	a	m	o	f	h	i	c	b	g	k	n	q	u	j	r	v	t	y	p	e	l	d	w	x	s
u	x	s	j	e	k	b	y	l	n	t	z	h	q	i	d	r	v	g	a	c	f	w	m	o	p
x	d	p	z	u	j	l	q	t	s	n	c	m	g	f	y	e	k	w	i	o	h	v	a	b	r
y	c	i	q	r	p	n	f	m	h	s	l	g	d	t	x	j	o	z	b	v	w	e	k	u	a
f	i	v	d	o	r	s	a	y	z	j	x	l	k	h	g	b	c	q	n	t	m	u	p	w	e
t	p	q	n	w	v	e	u	g	y	f	o	r	a	l	k	s	z	m	j	h	d	b	c	i	x
w	m	r	g	l	o	h	j	k	b	a	d	s	v	e	q	t	i	f	u	n	p	x	z	c	y
j	v	b	m	h	g	f	i	c	d	z	a	k	l	x	s	w	e	o	t	p	y	q	u	r	n
r	f	c	l	y	z	g	v	j	w	m	p	t	h	n	e	q	u	a	x	b	k	o	d	s	i
c	r	h	u	a	x	w	p	n	q	e	b	f	m	s	t	l	y	d	z	j	o	i	v	k	g
m	u	o	p	j	t	k	d	f	c	h	g	e	y	z	a	n	l	x	s	i	b	r	q	v	w

Trace: Down, Left, Up, Right, Down, Left, Down

Figure 14: A board, start location, and trace provided.

Given this information, there is a low probability of recovering the password. There are too many possibilities for a password. Having the trace does not give enough information to determine the password. Whether the password is a dictionary word or not has no effect in the difficulty of recovering the password.

Dictionary word

Modified CrossPassword

While our CrossPassword implementation relies on the security of the Latin Square, the modified version is no longer a Latin Square having a reduced size, 13×13, but still using all 26 letters as possible letters.

Intercepted Board

Unlike our previous implementation, the adversary gains information looking at even a single board. After looking at a few boards, the adversary may be able to determine all the letters of the password. If the adversary is able to intercept the board along with the start row or column, as sent by the server, then he can use the vulnerability that only $^{13}/_{26}$ of the letters can make up the first letter of the password by simply looking at the starting row. This already cuts down the first letter to ½ of all possible letters. By simply refreshing the page and getting a new board to see, the adversary can cut down the possible letters even more. He continues this process until there is only 1 letter that is always repeated across all boards in the start row /column; this letter is the first letter of the password. Now that the first letter is figured out, the adversary can repeat the process one letter at a time to calculate all letters in the password. For a random password, the adversary may not know when to stop, but if the password is a dictionary word, the adversary can stop at the end of a dictionary word.

Math to be added

Attack Pseudocode?

Intercepted Trace

Similar to our Original CrossPassword implementation, the adversary gains no information when multiple traces are presented other than the length of the password. They can then brute force the password knowing there are 2^{l-1} possible combinations.

Intercepted Board and Trace

Lastly, there is the possibility of having all information of the board, the start-location, and the trace. This equates to the information gained from have the board and its accompanying starting row, and the information gained from having the trace. The adversary can repeat the same process to obtain the password as described above for intercepting only the board and start row/column. Intercepting the trace comes with the added benefit that the adversary now knows the exact length of the password and knows exactly when to stop and when he has the password.

Math to be added: Same math as above?

Comment [MEP9]: What does the figure add here?

Comment [MEP10]: What would the attacker need to do?

Comment [MEP11]: Can stop because knows how many characters are in password by looking at a bunch of grids. Ie in a 9 character password there are 9 characters which appear 1x in a row and column always in every grid. Certainly possible.

Comment [MEP12]: Cuts down on the # of possibilities, requiring less grids (how many less?)

GrIDsure

There are 25^4 possible combinations of length-4 patterns.

However, there are 10^4 possible inputs to return.

We cannot exactly map one "trace" from the grid to the squares that the user selected because there are 25 grid locations but only 10 digits. Thus each digit will be in the grid an average of 2.5 times.

How many Grids and answers do we need?

Since each item is on the grid an average of 2.5 times there are 2.5^4 possible underlying grids for each given trace. That means that with only 1 observation, an attacker as a 1 in 39 chance of guessing the original box pattern correctly.

If the attacker has two observations, then there is a very small chance (how small) that more than two of the 39 possible grid patterns overlap. There will be our 1 target overlap, but a very, very small chance of accidential overlap.

Can you check this?

P(a PIN matches a particular random seq) = $\frac{39}{25^4}$

P(2 PIN matches a particular random seq) = $(\frac{39}{25^4})^2 = 9*10^{-9}$ randomly

(how do the math?)

Other Factors

We evaluate each system according to the criteria set out in The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes.[3]

Improvements to Criteria

Resilient-to-Physical Observations Category

We think that the **Resilient-to-Physical Observations** category should be split in two: **casual observation** and **video observation**. Casual observation is if an attacker is just able to watch the user enter their password once. This is feasible for short passwords and/or if the user types slow. An attacker can see which keys are hit on the keyboard. This is especially true if the user types slowly, has a short, and/or easily remember-able password.

However, the attacker seeing the user trace out the password on the grid once would have trouble remembering the entire grid, preventing the total loss of the password scheme. For that specific

[3] Bonneau, Joseph, Cormac Herley, Paul C. van Oorschot, Frank Stajanoy. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. University of Cambridge; Microsoft Research; Carleton University; University of Cambridge. Proc. IEEE Symp. on Security and Privacy 2012 ("Oakland 2012"). http://css.csail.mit.edu/6.858/2012/readings/passwords.pdf

Make Mine a Quadruple: Strengthening the Security of Graphical One-Time PIN authentication

Ravi Jhawar, Philip Inglesant, Nicolas Courtois, M. Angela Sasse
Dept. of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
ravi.jhawar.09@ucl.ac.uk, p.inglesant@ed.ac.uk, {n.courtois, a.sasse}@cs.ucl.ac.uk

***Abstract*—Secure and reliable authentication is an essential prerequisite for many online systems, yet achieving this in a way which is acceptable to customers remains a challenge. GrIDsure, a one-time PIN scheme using random grids and personal patterns, has been proposed as a way to overcome some of these challenges. We present an analytical study which demonstrates that GrIDsure in its current form is vulnerable to interception. To strengthen the scheme, we propose a way to fortify GrIDsure against Man-in-the-Middle attacks through (i) an additional secret transmitted out-of-band and (ii) multiple patterns. Since the need to recall multiple patterns increases user workload, we evaluated user performance with multiple captures with 26 participants making 15 authentication attempts each over a 3-week period. In contrast with other research into the use of multiple graphical passwords, we find no significant difference in the usability of GrIDsure with single and with multiple patterns.**

***Index Terms*—Graphical Passwords, one-time PINs, Usable Security, Man-in-the-Middle, Entropy, GrIDsure**

I. Introduction

Secure and usable authentication remains a challenge for most information systems. Despite other forms of authentication, such as biometrics, being developed, knowledge-based authentication through passwords and PINs are very widely used. However, Users have well-documented problems recalling text-based passwords or Personal Identification Numbers (PINs) [1], [14]; as a result and tend to choose predictable values [16], [29] or resort to other potentially unsafe practices.

PINs, in particular, have a low set of possible values, and users tend to select from an even smaller set of choices - to increase memorability, they either choose significant dates or use simple sequences of numbers such as 1248 [24]; for example, [21] reported a study which finds that 18% of users chose their birthdays as PINs. Given that PINs can be captured through attacks such as key-logging and shoulder-surfing, they are at risk of being compromised [12], [16]. Security could be improved through the use of one-time PINs; however, ever since their first use as far back as World War II they have been known to be difficult to use [18] and are not practical in many situations.

Graphical password schemes [8], [10], [15], [19], [28] have been proposed as more memorable alternatives to textual passwords, using the recognised ability of human memory to recall (or in some schemes, to recognise) pictures or shapes rather than text.

In this paper, we analyze the security of GrIDsure [13], a patented authentication scheme which combines a graphical, shape-based password with a one-time PIN, without requiring special hardware. GrIDsure has been applied to Microsoft IAG and UAG, Windows or Active Directory login - optionally as part of 2-factor authentication - document authorisation and signing, and has been implemented as an authentication option on a smartcard.

The statistical security of GrIDsure has been investigated by Weber [27], who concluded that GrIDsure is at least as secure as a static PIN, and, for threats such as shoulder-surfing provides far greater security. In contrast, Bond [6] argued that this scheme is no more secure than a standard PIN because users are likely to choose from a limited subset of predictable patterns.

Brostoff et al. [7] found that users did indeed chose predictable patterns unless they were instructed to pick less predictable ones, and concluded that Bond's analysis affected security in some usage scenarios, but not others. They recommended its use as a second factor authentication where the capture of both one-time PIN and grid is unlikely such as at Point-of-Sale. The authors argued that user performance with GrIDsure warranted further examination, and whether the security issues could be addressed through modifications.

In this paper, we exactly do that: with a detailed analysis of the security of GrIDsure from a new empirical work, we suggest an enhanced system to overcome the security issues identified in previous studies. The aim of our effort was to see if it is possible to increase the security of the system without reducing its usability. We therefore conducted a usability study on a prototype version of the new system and obtained encouraging results.

The remainder of this paper proceeds as follows. After describing the GrIDsure scheme and summarising previous studies on its security and usability, we demonstrate that GrIDsure as it stands is not resistant to intercepted communications. Exploiting the commonly used computer security concepts, we identify several enhancements which provide effective resistance for GrIDsure against Man-in-the-Middle attacks. We report the results of the usability of these enhancements based on our evaluation, and find that there is no significant difference in recall reliability between the original GrIDsure and our enhanced design.

978-1-4577-0460-4/11/$26.00 ©2011 IEEE

II. Introduction to GrIDsure

Graphical passwords are more easy to use than passwords and PINs because they offer cued (rather than unaided) recall and this makes them particularly well suited for infrequent authentication [3], [4], [23]. GrIDsure uses graphical scheme to generate a one-time PIN which users read off and enter into another application or device. It is essentially an example of a graphical password scheme and effectively a combination of both, a graphical and PIN authentication scheme.

With GrIDsure, the user has to remember a pattern rather than a passcode or a complex password. It works in two basic steps:

1) Registering Personal Identification Pattern (PIP)
   The user has to choose a pattern - a shape and sequence of squares on the grid, and register the pattern with username or account. The pattern can be of any length - e.g. 4 to replace a 4-digit PIN - and any shape that the user finds easy to remember. Note that the *order* of the chosen squares is significant.
   For enrollment, a grid with non-repeating characters spread in a random fashion can be used, where the user enters the characters which correspond to his or her chosen pattern.
2) Using the Personal Identification Pattern
   A grid with random numbers in each cell is displayed to the participant when he uses the system. The user then has to enter the numbers that correspond to his registered pattern as his one-time PIN.
   An example of a pattern on a random grid is shown in Fig. 1 (of course, in real use the cells are not shaded orange).

In principle, GrIDsure can be implemented on a grid of any reasonable size or shape; for the purposes of this study, however, we consider only the common use of a grid of 5X5 cells from which users choose ordered patterns of 4 cells and on which re-use of cells is allowed.

On authentication, if the digits 0-9 are used on a 5X5 grid, there will be some repeated numbers, and this is an important feature of GrIDsure security.

The system affords some protection from observation and replay attacks because *(i)* although a user's pattern is constant, the grid is randomised with each use, so the resulting PIN will be different each time; and *(ii)* there is always more than one possible pattern, on the randomised grid, that could have produced an observed PIN.

III. Related work

A. Graphical Passwords

From a usability point of view, the key advantage of graphical passwords over traditional knowledge-based authentication schemes is that they can offer cued, rather than unaided recall. Human memory performance with cued recall is significantly better than for unaided recall, particularly with infrequent usage [23]. Another advantage of graphical passwords is that psychology research has consistently found that pictures are recalled more readily than concrete words, and concrete words more readily than abstract [17].

8	4	5	9	1
9	5	4	0	2
0	2	8	3	7
3	3	7	9	6
7	6	8	1	7

Fig. 1: Entering a PIN (the cells are shaded for illustration only): 3, 6, 7, 3

Graphical passwords schemes, such as *PassPoints*, offer *cued recall*, typically involving users in recalling a specific target on an image, as presented in [3], [28]. Of the recognition-based schemes, perhaps the best-known graphical authentication scheme is PassFaces™ [8], [20]. The most similar to GrIDsure is the recall-based Background Draw-a-Secret (BDAS) [10] scheme.

Graphical password schemes can produce high levels of maximum theoretical entropy; a DAS pattern in which the total length of the strokes is 11, for example, has a raw entropy of around 53 bits [15]. However, the number of "memorable" DAS patterns is considerably lower than this, as [15] show, and if "memorable" is assumed to mean "symmetric" then the size is lower still [26].

Unfortunately, if users are permitted to choose their own passwords, graphical passwords in general can end up being weaker than textual passwords because users choose predictable credentials to improve memorability. For instance, with PassFaces™, users prefer certain types of faces - what Monrose and Reiter termed as "beauty bias" [19]. On the other hand, there is evidence that, in certain configurations graphical passwords can be less vulnerable to shoulder-surfing than strong textual passwords [25].

B. Existing Research on GrIDsure

Weber [27] performed an analysis of the *statistical* security of GrIDsure.

Assuming a 4-cell pattern, the probability of randomly guessing the correct PIN by simply typing a PIN is 0.0001, as for any other 4-digit PIN. However, an attacker can gain a higher probability of success by entering the PIN that corresponds to a randomly chosen pattern. The probability of guessing the correct PIN in this way is 0.000342102; this is higher than that from simply guessing the PIN because not all PINs occur in the grid with the same probability [27].

This is a key point in the consideration of GrIDsure security, because the probability of guessing a PIN generated from a secret pattern - not of guessing the secret pattern itself, but a PIN which matches it - is greater than the probability of guessing a 4-digit static PIN. The additional security claimed for GrIDsure therefore rests on its resistance to capture of

the transaction, together with the assertion that, unlike a static PIN, successful authorisation using a guessing method does not compromise the secret pattern.

The guessing probability can be minimised by choosing a grid calculated so that each digit appears as near as possible, an equal number of times, rather than strictly randomly chosen across the set of possible digits. For example, using digits 0-9, 5 digits appear exactly 3 times each and other 5 appear twice. This is called a "balanced grid". On a balanced 5X5 grid, the probability of guessing a correct PIN by entering a random pattern is 0.000116986 [27].

While a balanced grid makes random-guessing more difficult, it increases the risk from intercepted communications; having captured a PIN and grid, it is generally easier for an attacker to reverse-engineer the pattern with a balanced than with a random grid. We expand on this point later from our empirical work.

GrIDsure has been found to be easy to learn and the recall of patterns is acceptably reliable; however, as with other password schemes, the effective pattern space is far smaller than the maximum possible [7]. However, to understand the actual pattern space, simple assumptions are not sufficient [6]; as well as the shape, the order of cells and placement on the grid are important factors distinguishing between patterns. Although there are common patterns, these do not all occur with similar frequency [6], [7]. Brostoff et al. have developed a taxonomy of patterns, and our current work builds on this.

IV. Empirical Study of GrIDsure Security

We now re-consider the security and usability of GrIDsure from our empirical work. We are able to make an early estimate of the entropy of the GrIDsure's pattern space and give a far more thorough analysis of the risks from multiple captures than the rough figure of "2 in most cases" as suggested by Bond [6].

A. The Actual Entropy of GrIDsure

The maximum entropy of the possible pattern space for a 5X5 grid, from which users choose patterns of 4-cells is $\log_2(25^4) = \log_2(390625) = 18.5754$. This is considerably less than the 52 bits of entropy of a random 8-character password from a 95-character set, but comparable with a Draw-a-Secret password of length 4 strokes (which would be a very simple DAS password) [15].

However, the entropy of patterns actually chosen is lower than the theoretical entropy of the grid. From the patterns chosen by participants in our study described in section 6, we calculate a lower bound to the entropy, based on the calculation for a balanced estimator of the Shannon entropy:

$$\hat{H}_S^{bal} = \frac{1}{N+2}\sum_{i=1}^{M}\left[(n_i+1)\sum_{j=n_i+2}^{N+2}\frac{1}{j}\right]$$

from [5], where M is the number of patterns and N is the sample size. In our sample of 140 there were 102 distinct patterns, of which 78 were chosen once, 15 twice, 7 chosen 3 times, and 1 each 5 and 6 times. This gives a low entropy of 6.56, which suggests that GrIDsure may be rather easier to guess than it might first appear.

B. Resistance of GrIDsure to Interception

A capture of both PIN and Grid is possible in a Man-in-the-Middle (MiM) or shoulder-surfing attack. For shoulder-surfing, it is unlikely that an observer would be able to memorise the Grid at the same time as observing a PIN, but video recording would make this vulnerable. MiM could also be effectively carried out in the form of malware on the user's computer or a fake "Phishing" website. In this paper, we use MiM to refer to any situation where an attacker can capture both the grid and the user's PIN, and is therefore relevant to most systems even where there is reasonable security on the transmission channels.

In the case of traditional PINs or passwords, a single capture can be used by the attacker. In this limited sense, GrIDsure is an advance on traditional PINs. When GrIDsure is used as the authentication mechanism, the MiM can see the grid that the server sends to the user, keep a copy of it or change it and forward it to the user. In the same way, he can look at the user's response containing the one-time PIN which the user has read from the grid, and forward it to the server; he then has a copy of both the grid and the PIN.

Weber [27] shows that with a 4-cell pattern on a 5X5 balanced grid, an attacker can find on average 45.6976 patterns for each entered PIN. This seems like a reasonable improvement over a static PIN, particularly if, as is usual, an account or card is blocked after a number of consecutive authentication failures. But what if an attacker is able to make multiple captures? In the case of a MiM, this is realistic; if a communication channel has been intercepted, the intercept is likely to remain in place. In the following section, we show that if the MiM can successfully capture the grid and user response on multiple occasions, reverse engineering will rapidly reduce the possibilities to 1 pattern.

1) Multiple Captures: An illustration: As an illustration, consider the case in which an attacker successfully captures the first grid displayed in Fig. 2 and the corresponding user response (one-time PIN) captured is {3, 9, 0, 5}.

In this grid, 5 digits 1, 2, 7, 8 and 0 repeat twice and the other five digits are repeated three times; the grid is a balanced grid. From the user's response {3, 9, 0, 5}, digits 3, 9 and 5 occur three times in the grid and 0 occurs twice; thus the number of possible patterns that could correspond to the first grid with {3, 9, 0, 5} PIN is $3^3 * 2^1 = 27 * 2 = 54$.

To aid in matching the patterns, the adversary considers a grid numbered 1 to 25, left to right, top to bottom, as a reference grid to compare all distinct patterns. Using the reference grid, it is now possible to construct a list of all 54 candidate patterns. Examples of these patterns, numbered as in the reference grid, are {9, 12, 7, 6} and {21, 25, 23, 24} ...and so on up to 54.

Now, suppose the adversary captures the second grid and the corresponding PIN. In this case, suppose the user enters his PIN as {9, 7, 0, 5}. It so happens that the number of possible

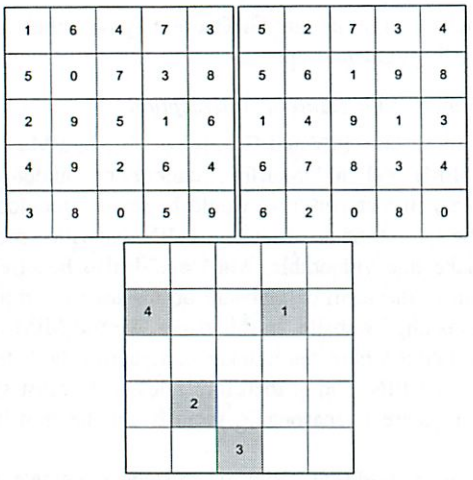


Fig. 2: First and Second captured grids in this example; below: Chosen pattern

patterns in this case is only $2^4 = 16$. As for the previous capture, the attacker lists all possible patterns for the second grid that gives the PIN {9, 7, 0, 5} using the reference grid; for example {13, 17, 23, 1}, {9, 3, 25, 6} ...and so on up to 16.

We now have two sets of possible patterns - one from each grid making use of the user's input. Comparing both sets of patterns, retaining only those patterns from the first set that have at least one matching entry in the second set, only a small number of possible patterns remains. In fact, in this case there is only one pattern matching both the sets, so the pattern has been reverse-engineered with only two captures; it is {9, 17, 23, 6} (see Fig. 2).

2) Emulating MiM Captures Programmatically: The previous section provided an illustration, but to understand the real risks, we have investigated the process of multiple captures empirically or mathematically. We chose to use a simple Monte Carlo technique to build grids programmatically and "capture" the PIN and grid in order to find the number of captures needed to reverse-engineer a pattern with certainty.

Having "captured" a PIN and grid, the program then simulates further captures of the same pattern (the grid and the corresponding PIN are obviously different), each time generating the set of patterns corresponding to the captured PIN and grid. In each iteration, the program also matches the patterns from the previously generated set, as described in the previous section.

The average number of patterns that match a captured PIN i.e. those which match all captured patterns in a trial are given in Table 1. Note that the average number of matches in a capture rises after 4 captures, but there are few trials that actually reach this number - most patterns have been found with fewer captures. We do not show the figures for later captures, for reasons of space. As expected, the average number of matching patterns in a capture from a random grid is far higher than from a balanced grid; this makes the attacker's job harder in terms of reverse-engineering the pattern but it is easier to make a successful random guess.

TABLE I: Numbers of matches after multiple captures

Iteration	Average matches	
	balanced grid	random grid
1	45.7019	146.6415
2	1.5605	2.5489
3	1.1003	1.1936
4	1.0874	1.1231
5	1.0910	1.1306

TABLE II: Captures to reverse-engineer a pattern

Found after captures	balanced grid	random grid
1	0	129
2	675282	422230
3	299699	496400
4	23304	72871
5	1599	7527
6	108	770
7	6	69
8	3	4

TABLE III: Expected and observed probabilities of matching patterns per PIN

Number of matches	Calculate Probability	Observed in 1000000 trials
16	0.0256	25661
24	0.1536	153611
36	0.3456	344925
54	0.3456	345995
81	0.1296	129808

From Table 2 also observe that, although there are a few reverse-engineered patterns using a random grid on the first capture while none with a balanced grid, the number of patterns that can be reverse-engineered with a random grid (42.2%) is far fewer when compared to a balanced grid (67.5%) with only 2 captures. The number of captures required with a random grid are higher from third and more captures.

By running our program over 1000000 simulated attacks, we obtained the following results for balanced and random grids:
Average number of captures to reverse-engineer a pattern:
Balanced grid: 2.3516; Random grid: 2.6680
Maximum: Balanced grid: 8; Random grid: 8

We already know, the expected number of PINs with patterns which match a PIN entered by a user, from Weber's [27] work; from our program, we are also able to make an estimate of the probable number of matches on multiple captures. After running our program for 1000000 trials, we generated a mean of the number of matching patterns. At the first capture, for a balanced grid this was 45.7019, similar to the expected value derived theoretically by Weber [27]. The different numbers of possible matches occur with different frequencies. The probabilities for each possible number of matches, which we have calculated using Weber's method of "templates" of pattern types, and observed in our simulation,

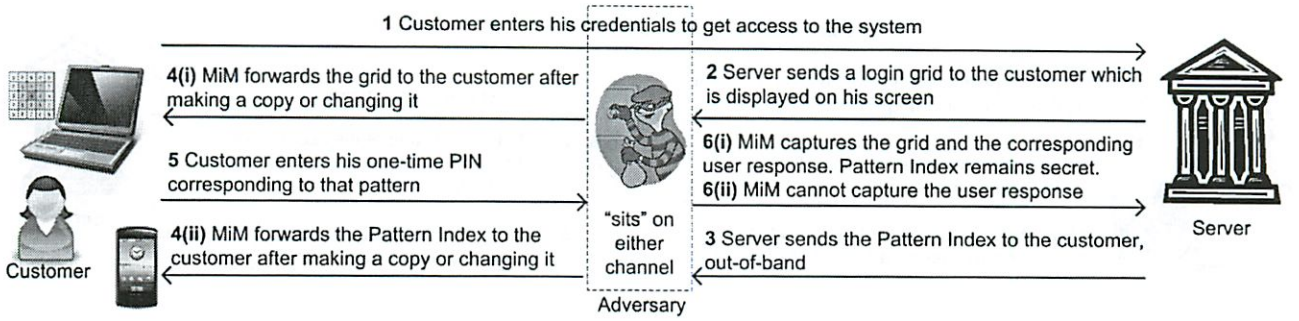


Fig. 3: Sending pattern index OOB when there is a risk of MiM attacks

are shown in Table 3. The closeness of the observed figures to their expected values indicates that our program is operating correctly. Note that we generate the "user's" pattern randomly for each trial, since not all patterns are equally easy to reverse-engineer.

So far, we have only modeled an attacker who assumes that all matching patterns are equally probable. If an attacker guesses patterns which are known to occur with higher probability, part of the subject of our further study, then the number of captures needed to reverse-engineer a pattern, which is already small, would be reduced further.

V. Fortifying GrIDsure against MiM

We have shown that GrIDSure is not resistant to MiM-type attacks since *(i)* patterns can be cracked with only a small number of captures and *(i)* the actual entropy is much lower than the theoretical entropy. In this section we present our modified system that greatly increase the resistance of GrIDsure to MiM and similar attacks.

A. Enhancement 1

In our proposed enhancement, Users choose and register multiple (different) patterns with his account/username, as shown in Fig. 4; for clarity, we show the grid in alphabetic order, although in actual implementation it is preferable for the enrollment grid to be ordered randomly, to prevent users from using guessable words as a form of pattern. Although, in principle, a system can implement our solution using any number of patterns, use of 4 patterns seem to be a reasonable balance considering that average users are now registered to more than 20 different accounts and have difficulties in managing their credentials [11].

Each time the user tries to login using our proposed system (GS4), he is informed which one among his registered patterns to use (the "pattern index") for successful authentication, using an Out-Of-Band (OOB) technique like sending an SMS to the user's mobile phone, as shown in Fig. 3.

With the use of GS4, unable to intercept the OOB channel, the attacker has no idea against which pattern a capture is to be matched. Simply comparing multiple captures will no longer reduce rapidly to a single matching pattern with a small number of captures.

Suppose an attacker has captured two PIN and grid transactions by intercepting the channel where the user enters his login details (steps 4(i) and 6(i) in Fig. 3). Of course, if the pattern index is the same for both captures (although the attacker cannot know this), then there must be at least one pattern that matches on both grids. However, this might happen even if the two patterns are different. An attacker finding that two or more captures match one unknown pattern could guess that all of the captures correspond to one pattern, but he cannot be sure.

On the other hand, with a large enough set of captures, the attacker can certainly *reject* some of the potential matches; comparing captures resulting from patterns which correspond to different pattern index will rapidly reduce to zero matching patterns. However, to find these sets of "non-matching" captures, every capture has to be compared with every other capture, and this still does not provide any patterns which are known with certainty. Eventually, by eliminating these non-matches, the attacker can build up a set of probable patterns, but note that the attacker still cannot know the corresponding pattern indexes.

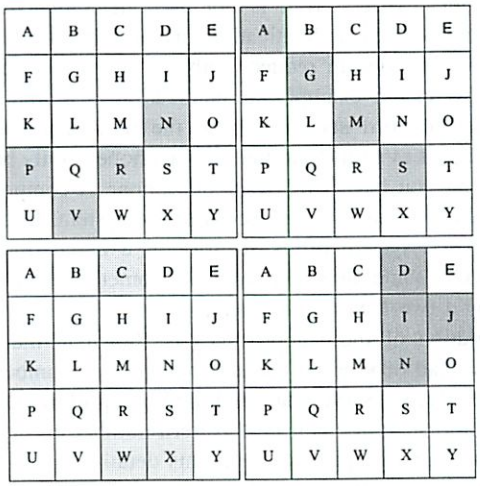


Fig. 4: Registering patterns in the proposed system. Pattern 1: PVRN Pattern 2: AGMS Pattern 3: KCWX Pattern 4: IDNJ

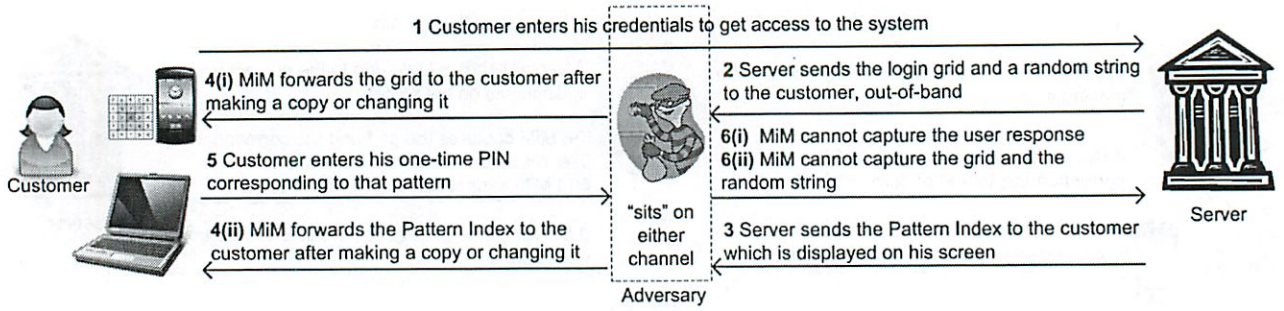


Fig. 5: Sending the login grid OOB when there is a risk of MiM attacks

If the pattern index generation does not follow a random distribution and is instead fabricated to ensure non-repetition of consecutive index sequences; assuming the system registers four patterns with each user account, the best that can be said is that the attacker can learn all the four patterns after 8 captures but still cannot know the pattern indexes. If all the patterns are known by the attacker, the probability of successfully entering a PIN in response to a challenge from the server is 0.25 on each attempt. In this case, clearly the number of attempts the server allows to login to the system becomes a critical parameter of consideration. If a user is allowed 3 attempts, the attacker has $\triangleq$ 0.75 probability of getting in to the system; this reduces to 0.25 if only one attempt is allowed, and if 4 attempts are permitted then the entire set of 4 patterns would be effectively compromised.

B. Enhancement 2

In a further enhancement involving Out-of-Band communications, another parameter, such as a random one time string, is also sent to the user's mobile phone along with expected pattern number. The user reproduces this string during login. Assuming that the attacker cannot control both communication channels, the attacker will never be able to login to the system.

It is worth a note that OOB cannot be used standalone to authenticate users. If it is not used in conjunction with a knowledge-based authentication mechanism like GS4, the security of the system reduces only to the physical security of the device. For example, authentication in a conventional username/password scheme where the user reproduces the password sent as an SMS to his mobile phone is only based on "something you have". If the mobile phone is compromised, the attacker can easily gain access to the system. In contrast, our proposed system offers security of a higher magnitude i.e. of both "something you have and something you know", providing a two-factor authentication.

A variation of these approaches is shown in the Fig. 5, where, instead of sending the expected pattern number to the mobile phone, the server can send the login grid to the mobile phone and display the expected pattern number on the screen where the user is expected to type his details.

In this case, it becomes almost impossible for the MiM to be able to capture the grid and the user response both. If the attacker "sits" on the communication channel between the server and the login terminal, he will learn only the one time PIN - i.e. the user's response and the expected pattern number (steps 4(ii) and 6(ii) in Fig. 5). If the attacker "sits" in the communication channel between the server and the mobile phone, he can see the grid which the server sends to the user, which in any case changes each time, but cannot capture the user's response (steps 4(i) and 6(i) in Fig. 5). This defeats the MiM attack, as it is highly unlikely that the attacker can control both the web and phone channel. The only drawback with this method is that the user must have a mobile phone that is capable of displaying the grid.

Since the server initiates the transmission, this channel is not at risk of being controlled by an attacker (even though it could itself be intercepted or overseen). However, the OOB channel could be any convenient form of electronic communication, which makes our proposal extremely flexible for use at, for example, an ATM.

VI. Usability of multiple patterns - an evaluation

Whilst the use of multiple patterns would fortify GrIDsure against MiM and shoulder-surfing attacks, this is not a practical solution unless it is possible for users to recall multiple patterns. Studies of other graphical authentication schemes have shown that adding a second graphical password significantly reduces the number of correct recalls - e.g. for Passfaces™ [26]. There is also evidence that multiple graphical passwords produce interference problems similar to those known in textual passwords [11]. Therefore, to investigate the usability of our proposals, we conducted a field trial evaluation, using the Authentication Performance Evaluation Tool (APET) online web-based tool described in [2].

A. Methodology

We used a Chi-squared test to decide the number the participants to be recruited for the study so as to obtain the best results. The results of the test suggested the value of N = 26. We recruited 30 people to allow for a 15% dropout rate.

30 participants from varying age groups and education levels were recruited over a three-week period. None of the participants had any previous experience of using GrIDsure.

Participants were divided into two groups. Each participant was assigned to make 15 login attempts over a three-week period during the experiment; this was done entirely using

TABLE IV: Reliability of recall of GrIDsure patterns

		Successful Login (%) >1 attempt	Successful Login (%) first attempt	Complete Failure(%)
Group A	GS1	2.77	95.83	1.38
	GS4	6.94	90.20	2.77
Group B	GS4	6.77	88.98	4.23
	GS1	1.50	98.50	0
Overall	GS1	2.22	97.03	0.74
	GS4	6.87	89.69	3.44

email and web. Participants in Group A used GrIDsure with one pattern (GS1) for first five logins and then used GrIDsure with four patterns (GS4) for the subsequent ten logins, whereas Group B used GS4 for the first ten logins and GS1 for the subsequent five. In this way, we avoided bias between the groups since each group used both designs, and we avoided bias within groups since the groups used the designs in different orders.

At the start of the trial, participants were sent an email requesting that they register their pattern(s) and then login using their first or only registered pattern (depending on the group) in the same session. The first three emails also included the instructions on the usage of the scheme, as initial training. All subsequent emails provided only those details necessary to login (no instructions). Participants were sent 4-5 emails a week over the three week experimentation period.

A well-known issue in the design of studies in usable authentication is that, in the real world, authentication is not users' primary task; they are using the authentication mechanism only to gain access to some service. For this reason, we devised a study in which the authentication was considered as the secondary, rather than the primary, task.

We used the *Barter World* scenario described in [2]. In this game, participants complete services - Gardening, Babysitting, Cleaning, or Teaching - for the community, and in return receive tokens for the appropriate working hours. (Participants did not actually have to perform the services, but they did have to log the hours worked to their personal account, protected by GrIDsure authentication, to claim their payment.)

As in [7], [11], participants were compensated with the exchange of tokens for gift certificates, at the rate of £1.33 for every successful login and £1.20 otherwise, giving a guaranteed minimum for participation up to a maximum of £20.

The community manager sent them an email when a barter task "had been completed". The email included a hyperlink and the pattern number (index) with which to authenticate, and an additional random string. If a user failed to authenticate within three attempts, the authentication server sent another email containing a hyperlink to the registered pattern; this simulates a real-world "password reset". If a participant failed to attempt authentication before midnight, they could no longer authenticate and log the claim.

The APET system [2] records *(i)* the time taken to login, *(ii)* number of login attempts, *(iii)* whether or not the attempt was successful, *(iv)* the IP address, and, if more than one attempt was required, *(v)* the PIN entered and what it should have been.

Following a successful GrIDsure authentication, participants entered a "claim code" consisting of the random string contained in the email. This implements the random out-of-band data suggested in section 5.2.

B. Results

As in Brostoff et al.'s study [7], user performance results are encouraging. All participants in Group A and 14/15 participants in Group B were able to login successfully.

In Group A, during the 75 usages (5 each by 15 participants) in the GS1 phase of evaluation, there were 3 occurrences of participants failing to respond to emails before midnight and hence of the request expiring. During the 150 usages - 10 each - of the GS4 phase, there were 6 occurrences of email expiry. In Group B, 2 participants discontinued the study. Of the 65 usages in the GS1 phase (which was completed after the GS4 phase for Group B) - 5 each by the remaining 13 participants - there were 2 occurrences of email expiry and 12 during the GS4 phase.

Excluding these non-attempted authentications, the results are shown in Table 4.

A failure rate of 3.44% would not be considered good in ordinary password use, but for an initial encounter with an unfamiliar mechanism, the performance is encouraging.

An important result, for the validating the usability of our proposal, is that there is no significant difference in user performance between GS1 and GS4.

We consider only 13 participants in each of groups A and B (13+13=26) as required by the Chi-squared test. This gives 127 of 128 successful logins within 3 attempts using GS1 and 237 of 246 successful using GS4.

Applying a χ^2 test, we find χ^2(df=1, Yates' correction) = 1.68, p=0.194. However, since the expected value of failed logins using GS1 is less than 5, which suggests that χ^2 may be unreliable, we also apply Fisher's exact test which gives a one-sided significance of p=0.091 (>0.05) i.e., not statistically significant, although it's not strong enough to say that GS4 is as easy as GS1. Intuitively, from our results, GS4 is at least a bit harder than GS1, but we can infer that we have not found it to be *significantly* less easy.

This finding is surprising - given that studies have found a clear interference effect for multiple passwords in other

graphical authentication schemes [11] - and encouraging. However, GrIDsure is quite different from the scheme used in Everitt et al.'s study, which was based on recognition of faces, similar to PassFaces™ [20]. In addition, factors such as frequency of use and length of use are known to have important impacts on the usability of passwords generally.

VII. Conclusions and Future Work

In this paper, we present empirical results which confirm the assertions of Brostoff et al. [7] and Bond [6], that GrIDsure is not resistant to multiple captures of the grid and PIN. We have also developed enhancements to GrIDsure which fortify it against Man-in-the-Middle and similar attacks.

From our user study we did not find the enhanced GrIDsure, with 4 personal patterns and Out-of-Band secrets, to be significantly less usable than simple GrIDsure.

Given the documented problems with interference between different graphical passwords just as for textual ones [11], it might be more usable to remember 4 patterns across a number of services than different patterns for each service. We agree with [7] that the re-use of patterns across different services is insecure in general. However, if it can be shown that our enhancements add sufficient security to enable the safe use of patterns across multiple services, it is possible that our solution will actually improve the overall usability of GrIDsure.

We have also completed a longer-term study of the usability, interference and memorability of GrIDsure with 4 patterns. Our results here have confirmed the importance of detailed research on actual user behaviour as the basis for the design of emulation experiments and estimates of actual entropy.

Therefore, our future work will provide a more detailed study of the effective entropy of GrIDsure based on the taxonomy of patterns which we are developing from our empirical studies. This taxonomy will also enable us to enhance our system using Monte Carlo technique, by emulating patterns actually chosen rather than random ones. We will also emulate an attacker who guesses similar patterns, or who uses the kinds of "clever" guessing methods suggested by Weber [27] by choosing the more frequently occurring digits from a grid. Finally, a more complex algorithm will enable us to emulate grid and PIN capture in our enhanced GrIDsure with 4 patterns.

References

[1] A. Adams, M. A. Sasse, and P. Lunt. Making Passwords Secure and Usable. *People and Computers XII: HCI 97*, 1997.

[2] A. Beautement and M. A. Sasse. Gathering Realistic Authentication Performance Data Through Field Trials. *Usable Security Experiment Reports (USER) Workshop, Symposium On Usable Privacy and Security*, 2010.

[3] R. Biddle, S. Chiasson, and P. C. van Oorschot. Graphical Passwords: Learning from the First Generation. *Technical Report TR-09-09*, 2009.

[4] R. Biddle, S. Chiasson, and P. C. van Oorschot. Graphical Passwords: Learning from the first twelve years. *ACM Computing Surveys*, 2011.

[5] J. A. Bonachela, H. Hinrichsen, and M. A. M. noz. Entropy estimates of small data sets. *Journal of Physics A: Mathematical and Theoretical*, 41(20):1–9, 2008.

[6] M. Bond. Comments on Gridsure Authentication. 2008.

[7] S. Brostoff, P. G. Inglesant, and M. A. Sasse. Evaluating the usability and security of a graphical one-time PIN system. *Conference on Human Computer Interaction BCS*, 2010.

[8] S. Brostoff and M. A. Sasse. Are Passfaces more usable than passwords? A field trial investigation. *HCI 2000 - People and Computers XIV - Usability or Else! BCS*, 2000.

[9] J. Cohen. Quantitative Methods in Psychology, A Power Primer. *New York University*.

[10] P. Dunphy and J. Yan. Do Background Images Improve Draw a Secret Graphical Passwords? *Conference on Computer and Communications Security*, pages 36–47, 2007.

[11] K. M. Everitt, T. Bragin, J. Fogarty, and T. Kohno. A Comprehensive Study of Frequency, Interference, and Training of Multiple Graphical Passwords. *27th International Conference on Human factors in Computing Systems*, 2009.

[12] D. Florêncio and C. Harley. A Large-Scale Study of Web Password Habits. *Proceedings of WWW 2007 (Banff, Alberta, Canada*, May 2007.

[13] GrIDsureLimited. http://www.gridsure.com.

[14] P. G. Inglesant and M. A. Sasse. The True Cost of Unusable Password Policies: Password Use in the Wild. *28th International Conference on Human Factors in Computing Systems (CHI 2010)*, 2010.

[15] I. Jermyn, A. Mayer, F. Monrose, M. K. Reiter, and A. D. Rubin. The Design and Analysis of Graphical Passwords. *8th USENIX Security Symposium*, 1999.

[16] D. V. Klein. Foiling the Cracker: A Survey of, and Improvements to, Password Security. *Second USENIX Workshop on Security*, pages 5 – 14, 1990.

[17] S. A. Madigan. Picture memory. *Imagery, Memory, and Cognition, Yuille, J. C. (ed.)*, 1983.

[18] L. Marks. Between Silk and Cyanide: A Codemaker's Story 1941-1945. *HarperCollins, London, UK*, 2000.

[19] F. Monrose and M. K. Reiter. Graphical Passwords. Chapter 9 in Security and Usability: Designing Secure Systems That People Can Use, Cranor, L. F. and Garfinkle, S. (eds.), O'Reilly, Sebastopol, CA, USA; Cambridge, UK. pages 161 – 179, 2005.

[20] passfaces.com. http://www.passfaces.com.

[21] One in five use birthday as PIN number. http://www.telegraph.co.uk/finance/personalfinance/borrowing/creditcards/8089674/Oneinfiveusebirthdayas PINnumber.html.

[22] A. Polyviou. The impact of interference and frequency of use on the performance of three authentication mechanisms. *Masters thesis (unpublished), University College London*, 2010.

[23] M. A. Sasse, S. Brostoff, and D. Weirich. Transforming the 'Weakest Link' - a Human/Computer Interation Approach to Usable and Effective Security. *BT Technology Journal*, 19, July 2001.

[24] E. M. Tamil, A. H. Othman, S. A. Z. Abidin, M. Y. I. Idris, and O. Zakaria. Password Practices: A Study on Attitudes towards Password Usage among Undergraduate Students in Klang Valley, Malaysia. *Journal of Advancement of Science & Arts*, 3:37–42, 2007.

[25] F. Tari, A. A. Ozok, and S. H. Holden. Comparison of Perceived and Real Shoulder surfing Risks between Alphanumeric and Graphical Passwords. *Symposium on Usable Privacy and Security (SOUPS 06), ACM Press*, 2010.

[26] J. Thorpe and P. C. van Oorschot. Graphical Dictionaries and the Memorable Space of Graphical Passwords. *13th USENIX Security Symposium*, 2004.

[27] R. Weber. The Statistical Security of GrIDsure.

[28] S. Wiedenbeck, J. Waters, J. C. Birget, A. Brodskiy, and N. Memon. PassPoints: Design and longitudinal evaluation of a graphical password system. *International Journal of Human-Computer Studies*, (63):102 – 107, 2005.

[29] M. Zviran and W. J. Haga. Password Security: An Empirical Study. *Journal of Management Information Systems*, 4(15):161–185, 1999.

Comments on Gridsure Authentication

This document is an edited version of notes taken to provide feedback to Gridsure after an approximately two hour meeting on Friday 8th February 2008, and based on past presentation material on the scheme seen in mid 2007.

Please note that these comments are *selective* feedback on particular issues I found with the Gridsure scheme, which can aid other analysts in continuing this work. This document is not intended to be a fully representative or balanced appraisal of the scheme.

Mike Bond, 27th March 2008

Weber's Report

I dispute the worth of Professor Weber's analysis[1]. Whilst his mathematical calculations in themselves I'm sure are flawless, there are a number of tacit assumptions made that undermine its meaningfulness, mainly about the psychology of choice of patterns. Weber first selects a set of "likely to be chosen" shapes, including lines, ticks and boxes. On what basis is it argued that users are likely to pick these shapes? Intuitively we might want to believe that squares, lines, ticks are all common, but it psychology results often defy intuition and need to be properly researched. Secondly, Weber assumes that all alignments of common shapes are equally likely to be chosen, for example that a four digit line running from left to right could start from the second column as well as the first. Beyond this, the combinatorics clearly say nothing about the relative likelihood of different patterns actually being chosen. So if we were to accept his assertion that there are 11,640 common patterns on the grid, it still is of huge significance that some of these patterns are more common than others. So the results of this report rely on psychology assumptions that Weber has not justified, and totally ignores the wider issue of relative probabilities of different shapes.

My suspicion is that the practical entropy of Gridsure patterns will be at least as low as that for PINs (way less than 10,000 combinations because PINs involving dates and years (e.g. 1984) are probably more common). Now, for PINs this can easily be rectified by issuing initial PINs rather than encouraging cardholders to choose their own. The advantages of Gridsure are eroded if the user cannot select his or her own pattern, and the results of the usability study conducted by Sasse are no longer applicable in this scenario.

Resistance of Challenge/Response to Pattern Recovery

During the meeting Gridsure stated that (according to Consult Hyperion), three "engineered grids" were required to determine a PIN with certainty. Below follows

[1] "GrIDsure – The mathematics of Patterns & Sequences", provided to me by Gridsure on 11/06/07

excerpts from my original analysis of June 2007, which Gridsure does not seem to have taken note of. Consider the following two engineered grids:

Challenge A	**Challenge B**
12345	11111
12345	22222
12345	33333
12345	44444
12345	55555

If a user is challenged with the following two patterns, then the pair of response codes together will leak the X and Y coordinates of each digit in the pattern. Clearly then a maximum of two challenge grids are required, not three as suggested by Consult Hyperion. Challenges A & B can have their digits permuted randomly, so long as the same transformation is applied to both A & B. The human eye will then not be able to detect the presence of a pattern. Furthermore, digits can be doubled up, so that pairs are used interchangeably. For example, the challenge grid A would become as follows:

Challenge A	**Challenge A (doubled up)**
12345	12895
12345	67345
12345	12890
12345	17345
12345	67345

Here the pairs are as follows: 1&6, 2&7, 3&8, 4&9, 5&0. The grid already appears much more random to the human eye, and remains just as usable for the attack; and this is before even the permutation step is undertaken.

Even better results could be achieved by using specially designed challenge grids based on empirical analysis of the common shapes and patterns chosen. The grid could be specially designed to make it as likely as possible that the pattern can be determined with a single challenge. For example, supposing we knew that straight lines either horizontally or vertically were 100 times more likely to be used than any other, then consider the following challenge grid.

Challenge C
12345
34567
56789
78901
32609

All possible length 4 horizontal and vertical straight lines are then uniquely coded:

1234, 3456, 5678, 7890, 3260	(horizontal, starting left most)
2345, 4567, 6789, 8901, 2609	(horizontal, leftmost + 1)
1357, 2468, 3579, 4680, 5791	(vertical, topmost)
3573, 4682, 5796, 6800, 7919	(vertical, topmost + 1)

This is just an example. With some care and attention, a very efficient single challenge grid aimed at exposing the most common shapes could be created.

So the standard mode of the gridsure system clearly is not strongly resistant to chosen challenges (such as might be deployed in a phishing attack). What then is the information leakage from response to a randomised grid challenge?

In a random challenge grid filled with digits 1 through 10, evenly distributed, we can expect every challenge to reduce (on average) the range of possible squares for each element of the pattern by a factor of 10 – namely a leak of 3.3 bits of information per digit responded to, or a total of 13.3 bits for a 4 digit response.

A fully random 4 digit pattern will have 25*24*23*22 combinations, equal to 18.2 bits of information (assuming non-repetition of the same square in pattern).

Clearly after two challenges, up to 26.6 bits of information have been revealed in the response, yet the maximal entropy of a pattern is 18.2 bits. Given the typical entropy of a pattern will be much lower, due some common patterns being much more appealing than others, it is likely that knowledge of both challenge grid and response from a single challenge will yield enough information to determine the pattern fully in a significant proportion of cases, and two challenge/response pairs will suffice in most cases. Thus if an eavesdropper is able to observe the challenge and response (e.g. with a mobile phone camera or fixed CCTV camera in a shop) or via screen capture in the case of malicious software on a PC, the pattern will quickly leak.

Though this analysis is far from complete, my opinion so far is that if the challenge can be seen, Gridsure is no more secure than a PIN, and possibly less so for the reasons described in other sections of this document.

Shoulder Surfing

The analysis of resistance to shoulder-surfing based on experiments with Children was inadequate. The learning curve and dynamics of teaching shoulder surfing are not known. Consider pick-pocketing – a criminal skill which requires some considerable level of practice to get good at. Yet we know it still can be done. Likewise shoulder surfers could specifically learn to determine patterns in a better way, probably with reference to common patterns. What we don't know is whether this is easy or difficult to learn, and it would be unwise to assume either. Note also that because the user always has to respond freshly with a different number to the challenge grid, the user will not be able to type the response number so quickly, and potentially not whilst also shielding it with their hand. This sort of protection is only likely to come into play upon repeated entry of the PIN. So one cannot assume that the response PIN is as well protected in the case of Gridsure (indeed it is proposed for accessibility purposes that certain respondents might read their response code aloud).

Compromised Terminals

The major current threat for PIN recovery in Point-of-Sale environment is not shoulder surfing (where Gridsure provides limited resistance), nor hidden cameras (where Gridsure resistance is even less as entire challenge and entry can be recorded for later review), but *compromised terminals*. This is where the Point-of-Sale terminal is sabotaged in order to record PIN and account details. The Gridsure scheme is no more resistant than PIN against sabotaged terminal, as the sabotaged terminal can record entire challenge and response (or indeed submit an engineered grid and then translate the response code from this grid to the response code for the grid received from the central server).

Multiple Entry Attempts

When the correct pattern cannot be determined with certainty (probability 1) from a challenge and response pair, it must be borne in mind that if there were several candidate patterns that could not be distinguished from one another, the user trying to authenticate will get (for example) three attempts to respond correctly. This means that if a challenge/response pair yields three possible patterns, then the attacker will still be able to respond correctly with certainty.

Side Channel Leakage of PIN

From early experimentation during the meeting, considering disjoint patterns or patterns with a change in direction (e.g. the tick), such as the following examples:

12345	12345
12345	12345
12345	12345
12345	12345

I noted that I hesitated during entry of the response code as I negotiated the gap or the change in direction of the sequence. If the time intervals between keypresses were monitored as well as the key presses themselves, this could yield extra "sidechannel" information about the nature of the pattern entered, which could help resolve between different possible combinations. Such timing attacks (and other sidechannels which act as windows on the mental processes of the secret holder) should be considered.

Writing down the Pattern

Some people are unable to remember PINs. It is conjectured that Gridsure patterns are easier to remember than PINs, though no evidence has been offered to this effect (I do note some of Sasse's references are broadly in support of this thesis however). Those who are unable to remember PINs often write them down, with the advice that they disguise their PIN, for instance as the area code of a telephone number. This means that if their wallet is stolen, the criminal will have to search carefully to try and recover the disguised copy of the PIN, and even then may not be successful. To accommodate those who do not wish to use Gridsure (but who are not sufficiently disabled as to actively reject its use e.g. some people use "chip & signature" cards

instead of chip&PIN), how might they record their correct Gridsure pattern in such a way that it can be easily concealed? This issue is unresolved, and as stated in the meeting, one should plan for a scheme to be resilient against disinterested, reticent and even sometimes totally self-destructive behaviour from users.

Screen Scraping, and Retrieving Challenge Grids from PCs

During the meeting Gridsure discussed how the scheme could be used in an online environment (for instance integrated with 3DSecure or VBV). Whilst Gridsure clearly provides no protection against phishing here (as engineered grids can be submitted), or against man-in-the-middle, it does apparently provide protection against keyboard logging viruses/worms/trojans.

Why do viruses not commonly scrape screens to retrieve password information? The answer is because the economics are not yet aligned for it to become necessary. There are easier and better ways at current to attack which do not require this technology. Yet the technology definitely exists and is demonstrably in the hands of the crooks as it is being used to recognise the text from "CAPTCHAs" – the distorted codes or phrases that one often has to re-type when signing up for a new account at a website. These are designed to resist automation by computers, but come under regularly attack, demonstrating that the crooks have the capability to perform sophisticated image processing in order to defeat security mechanisms.

With regards to screen scraping from Flash plugins or from "Silverfish", it may be true that current deployed screen scrapers have an issue with this, but this is not the same as saying that it cannot be done. As soon as the economics yields a reason to want to scrape from Flash, it will become possible. There are no significant technical barriers to attacker code running on a compromised machine reading all the screen information it likes.

Mobile Phone Gridsure

A variety of schemes were discussed where the Gridsure grid is rendered by a mobile phone, including methods where the challenge grid arrives in encrypted form via SMS, or where a challenge grid is constructed from pertinent transaction data such as destination and sort code. All of these schemes rely on the security of the mobile phone as an independent channel, and on the underlying cryptography. None of the detail of this proposed cryptography was presented in the meeting, so one cannot say either way if it would work or not. However it seems that this cryptography (if implemented successfully) would stand alone to make a formidable authentication mechanism, and the Gridsure code itself has rather little to add – only a substitution of PIN entry into the phone with generation of response code from challenge.

Memorability of Multiple Patterns

A study is recommended into the memorability of multiple patterns, although the idea of differently "cueing" the grid with framework patterns in order to evoke memory of a particular associated pattern is indeed clever, and a clear advantage over PIN prompts. This advantage should be stressed more strongly when comparing to PIN

entry. Whether these cue frameworks could be implemented effectively on black and white screens is a matter for further research too.

Summary of Threats and Protection

Threat	**Gridsure**	**PIN**
Shoulder surfing at POS/ATM	Partially resistant. Difficulty of shoulder surfing unknown	Partially resistant. Some studies of shoulder surfing performed.
Camera at POS/ATM	Not resistant	Not resistant
Sabotaged POS/ATM	Not resistant	Not resistant
Online, keyboard logger trojan	Resistant	Not resistant. However resistance achieved easily through PIN entry using drop-down boxes.
Online, Phishing (naïve clicking on emails)	Not resistant (engineered challenge grids)	Not resistant
Online, Phishing (Installed trojan misdirects user to bad site even though correct URL typed)	Not resistant	Not resistant
Shoulder surfing at PC logon	Resistant. Economics not in favour of colleageues learning to shoulder surf grids.	Partially resistant.
Physical keyboard logger attached to keyboard cable	Resistant.	Not resistant.

Conclusions

The Gridsure authentication mechanism remains largely unproven. Studies so far are flawed or taken out of context; my own initial studies indicate further weaknesses.

Many of the attacks discussed in this document rely upon Gridsure becoming a focus of attacks – for the economics to work – as it would indeed become were it used in a Point-of-Sale environment. Gridsure could well be more suitable for deployment in enterprise scenarios. Indeed it does provide protection against certain enterprise threats such as keyboard-cable keyloggers. Only if it achieves a large market share would it become economic to develop the attack methodologies properly.

Mike Bond

11th Feb 2008
(edit 27th March)

Math

12/14

SP

Sure attach as before

Yeah can elim letters ?
but

Were do start locating
└ actually less...

$$\left(\frac{1}{26}\right)^4$$

Possible char or avg

0 25
1 24
2 23

~~25~~
25 0

(2)

6,ID easier to figure out pw
but harder to brute force

---

trace
and what doesn't show up

6rid → gives 65 possibilites

2 grids 3
3 1 pure just trace

then add that many letters not shown

$\frac{6}{26}$ chance will show up

$\left(\frac{6}{26}\right)^2 = 5\%$

(3)

0 12

⋮

12 0

(wish I had more time to dig into prob)
↳ this is fun!

Prob.

Pick 1 item from bowl of 25

Prob Pick same item $= \frac{1}{25}$ (replace)

2 items

P pick same $\leq$ items $\frac{2}{25}$

3 $\frac{3}{25}$

Say

2/9

but this is 2nd chance

$\frac{4}{25} \cdot \frac{4}{1}$ get it

don't ye

$\frac{2}{25} + \frac{23}{25}$. hex - - -

Oh, girl
wish I had more file
or someone who knows this

(5)

12% → 9th user i

1st round

1st 12 shown
2nd 6
3
1,5

---

random up + down

Yeah can't get back underlying pw
even w/ 20 passwords

"But can ya know that ya always go
up or down."

Yeah!

Final 12/14 11:30P

CrossPassword: Novel Password Systems Where Enter Derivation of Password instead of Actual Password

6.858 Final Project

Michael Plasmeier <theplaz>
Jonathan Wang <jwang7>
Miguel Flores <mflores>

December 14, 2012

Motivation

The problem with many password systems is that users must type their entire, full password each time they log on. This makes the password vulnerable to key logging, shoulder surfing, and interception during transmission.

We explore systems in which the user does not enter their direct password, but a derivation of the password **which changes on each log in**. The user proves that he or she knows the password without subsequently ever providing the password itself.

ING Password Keyboard

A simple example is ING Direct's PIN pad. Under ING's system, the user enters the letters corresponding to their PIN instead of the PIN itself. The mapping between numbers and letters is randomly generated on every log in. This method does not survive an attack where the attacker has access to the mapping, but it does prevent simple keylogging.

Figure 1 ING's Pin Pad. The user enters the letters corresponding to their PIN in the box.

Answering Questions

One could answer questions about the password, instead of inputting the password itself. For example, say a user's password is "tennis ball." The system could prompt "what color is it?" The user would

respond "green." The next time the system could ask "what shape is it?" The user would respond "round." This way the user only transmits their actual object during registration, but never during log in.

Hashing a Response

In an ideal world, the user could prove to the server that it knew the secret by producing a cryptographic hash of the user's secret combined with a server-selected nonce.

$$Hash(Secret_{User}, Nonce_{Server})$$

However, people are not particularly good at being able to calculate cryptographic hashes in their heads, so we need to seek an alternate system.

Inspiration

Original Off the Grid

We were inspired by the "Off the Grid" system from the Gibson Research Corporation.[1] The "Off the Grid" proposal is designed to allow users to use a personal printed paper grid to encipher the domain name of the website they are currently on into a string of pseudo-random characters.

The Off the Grid system works entirely on the user's side. Websites do not need to do anything to support Off the Grid.

To use Off the Grid, the user first generates a grid from a grid-providing website such as https://www.grc.com/offthegrid.htm. This website generates a grid using client-side scripting (i.e. JavaScript) to generate the grid on the user's machine. The user then prints the grid onto a sheet of letter paper. At this point the Grid is offline and thus impossible to access by malware. As an alternative, there is at least one application for Android which produces and stores a grid; however, the grid is now accessible to malware on the Android phone which is able to defeat the inter-process sandboxing.

The grid that is generated is a Latin Square. A Latin Square is an $n \times n$ array filled with n different symbols, each occurring exactly once in each row and exactly once in each column.[2] The most famous Latin Square is the popular puzzle game Sudoku. (Note however, that we do not divide up the grid into 9 smaller 3x3 mini-squares in which each symbol must be unique). For example, here is a 11x11 Latin Square with 11 alphabetic characters: Normally, a 26x26 Latin Square is used.

g	e	a	m	o	n	z	k	i	r	c
k	a	n	c	m	z	o	r	g	i	e
n	k	c	z	a	m	r	g	o	e	i
z	o	i	a	n	g	e	c	r	k	m
m	r	z	n	g	a	k	i	e	c	o

[1] https://www.grc.com/offthegrid.htm and associated pages. Is still marked as "Work in Progress;" Retrieved 12/2/2012

[2] http://en.wikipedia.org/wiki/Latin_square

a	g	e	i	z	r	n	o	c	m	k
r	c	g	k	e	i	m	n	z	o	a
e	n	k	g	i	o	c	z	m	a	r
i	m	o	e	r	c	g	a	k	n	k
c	i	r	o	k	e	a	m	n	z	g
o	z	m	r	c	k	i	e	a	g	n

Figure 2 An 11x11 Latin Square; normally 26x26, but reduced in size here to save space.

Once the user has a grid, they use the grid to create or change the password for each website. The Off the Grid specification has a number of variants, but we will use the base variant described on the GRC website. The author recommends that each user adopt slight variations to the rules in order to increase security. To provide a consistent analysis, we assume the user ignores this suggestion.

In the Off the Grid specification, the user traces the name of the website twice to provide additional entropy. In the start of the first phase, the user always starts along the first row of the grid.

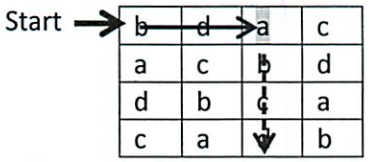


Figure 3

The user then traces out the first 6 characters of the domain name. 6 characters was chosen by the author to provide a 12 character password, which the author chose to balance ease of use with entropy. Again, a user may choose their own scheme. The user alternates between looking horizontally and vertically.

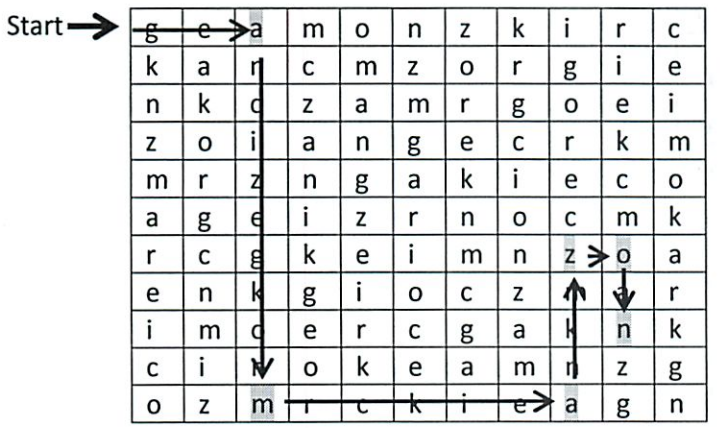
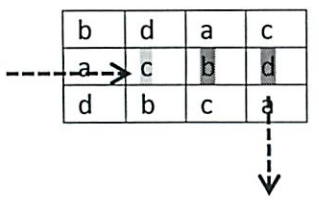


Figure 4

In the second phase, the user starts at the character that they ended with at the end of Phase 1. The user then selects two more characters from the grid in the same direction of travel. The user then appends those two characters to their password.



c	a	d	b

Figure 5 The user arrives at c traveling to the right. The user appends the next two characters "bd" to their password, and then continues up/down from the last character they read "d".

The user wraps around if their characters go off the grid.

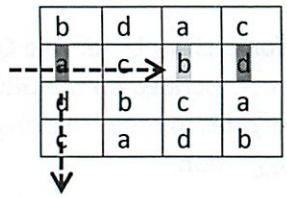


Figure 6 The user arrives at b traveling to the right. The user appends the next two characters to their password, wrapping around if they go off the edge of the grid. Here those characters are "da". The user then continues up/down from the last character "a".

For example here is Phase 2 of our Amazon example.

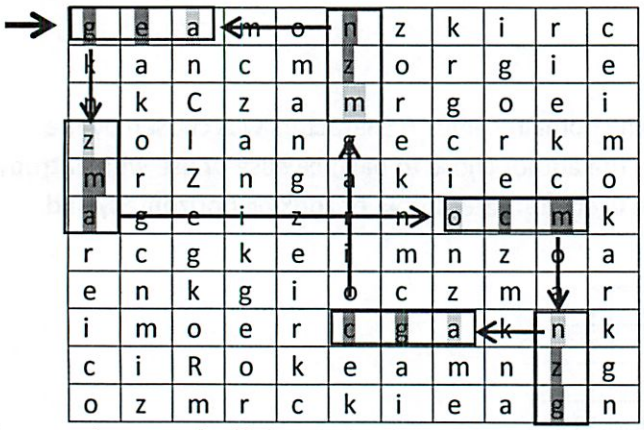


Figure 7 Phase 2 of Off the Grid. The password is "gaznegmacmzg"

Here are Phase 1 and Phase 2.

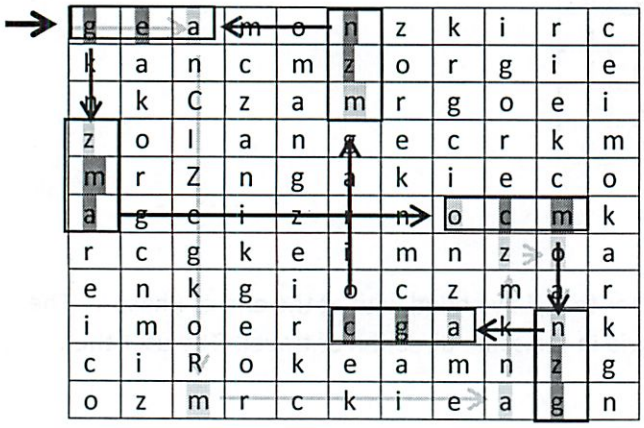


Figure 8 Phase 1 and 2 of Off the Grid.

To log in, the user retraces exactly the same steps as when creating a password. This means the password is exactly the same for each domain. This is an obvious requirement for a system designed to fit within the existing password infrastructure. However, we wanted to explore ideas in which the user does not enter the same password each time.

Description of System

CrossPassword

We wanted to design a system similar to the Off the Grid system, but where the password the user transmits over the network is different each time. With this system, the website presents the user with a grid and the user enters only a deviation of their password.

When the user creates an account, he provides his or her password to the webserver. The user may use characters from the lower case Latin alphabet [a...z]. The password may not have consecutive repeating characters, for example, "aardvark" has the repeating characters "aa" so it would not be allowed. The password is stored on the server such that the plain text can be accessed in order to verify the trace.

When the user logs in, the server randomly generates a 26x26 Latin square with the characters [a...z] called the *Grid*. The server also randomly selects a cell and a direction (either horizontal or vertical) as the *start location*. The server transmits this Grid to the user. The Grid and the start location are unique for each log in. The server stores the Grid and start location in temporary state and provides a pointer to this state called the *token* to the user. The user's browser returns the token to the server on each log in attempt.

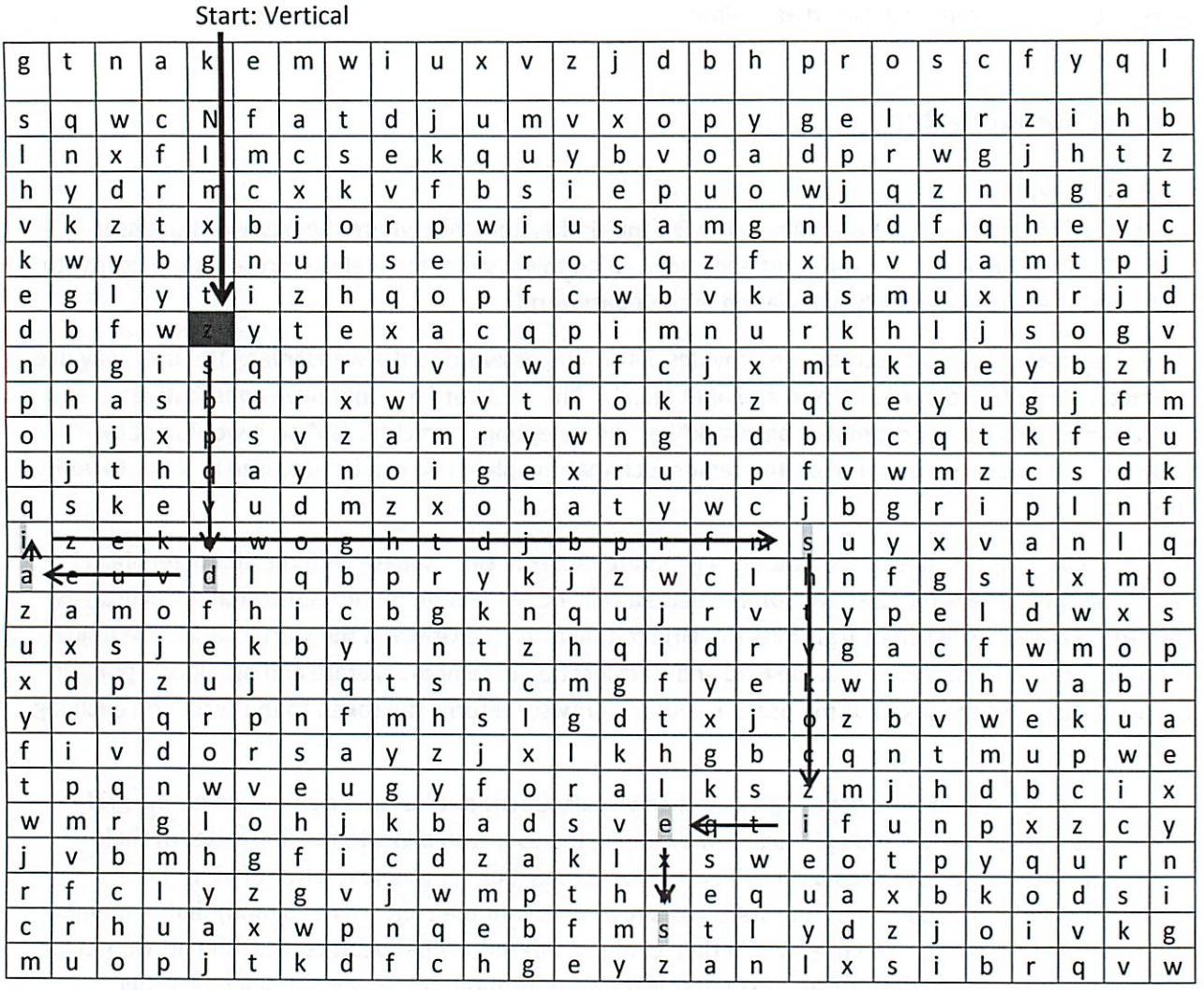
These are transmitted to the user. The user then visually traces out his or her password on the grid, alternating between rows and columns. For example, the user would locate the first letter of their password on the start row or column. The user would then look for the next letter of his or her password in either the column (if the start was a row) or row (if the start was a column) that contained the user's first character. The user would then continue alternating between vertical and horizontal for the length of their password. If the first letter is the start location, the server will select another random start location.

The user enters the directions (up, down, left, right) that they follow as they trace out their password. This is called the *trace* of the password. The trace and the token are sent back to the server.

The server verifies that the trace by replaying the trace and making sure the password letters match the provided trace.

The server will only accept 2 traces per token. If a user guesses incorrectly twice, the server will present the user with a new Grid and Start Location. The server will lock the account and the IP address after four incorrect tries until the user completes an email loop.

Example: entering the password Amazon with the 5th column as the start row/column. The grid, as well as the start location and direction, are randomly generated be the server for each log in.



g	t	n	a	k	e	m	w	i	u	x	v	z	j	d	b	h	p	r	o	s	c	f	y	q	l
s	q	w	c	N	f	a	t	d	j	u	m	v	x	o	p	y	g	e	l	k	r	z	i	h	b
l	n	x	f	l	m	c	s	e	k	q	u	y	b	v	o	a	d	p	r	w	g	j	h	t	z
h	y	d	r	m	c	x	k	v	f	b	s	i	e	p	u	o	w	j	q	z	n	l	g	a	t
v	k	z	t	x	b	j	o	r	p	w	i	u	s	a	m	g	n	l	d	f	q	h	e	y	c
k	w	y	b	g	n	u	l	s	e	i	r	o	c	q	z	f	x	h	v	d	a	m	t	p	j
e	g	l	y	t	i	z	h	q	o	p	f	c	w	b	v	k	a	s	m	u	x	n	r	j	d
d	b	f	w	z	y	t	e	x	a	c	q	p	i	m	n	u	r	k	h	l	j	s	o	g	v
n	o	g	i	s	q	p	r	u	v	l	w	d	f	c	j	x	m	t	k	a	e	y	b	z	h
p	h	a	s	b	d	r	x	w	l	v	t	n	o	k	i	z	q	c	e	y	u	g	j	f	m
o	l	j	x	p	s	v	z	a	m	r	y	w	n	g	h	d	b	i	c	q	t	k	f	e	u
b	j	t	h	q	a	y	n	o	i	g	e	x	r	u	l	p	f	v	w	m	z	c	s	d	k
q	s	k	e	y	u	d	m	z	x	o	h	a	t	y	w	c	j	b	g	r	i	p	l	n	f
i	z	e	k	w	o	g	h	t	d	j	b	p	r	f	m	s	u	y	x	v	a	n	l	q	
a	e	u	v	d	l	q	b	p	r	y	k	j	z	w	c	l	h	n	f	g	s	t	x	m	o
z	a	m	o	f	h	i	c	b	g	k	n	q	u	j	r	v	t	y	p	e	l	d	w	x	s
u	x	s	j	e	k	b	y	l	n	t	z	h	q	i	d	r	y	g	a	c	f	w	m	o	p
x	d	p	z	u	j	l	q	t	s	n	c	m	g	f	y	e	k	w	i	o	h	v	a	b	r
y	c	i	q	r	p	n	f	m	h	s	l	g	d	t	x	j	o	z	b	v	w	e	k	u	a
f	i	v	d	o	r	s	a	y	z	j	x	l	k	h	g	b	c	q	n	t	m	u	p	w	e
t	p	q	n	w	v	e	u	g	y	f	o	r	a	l	k	s	z	m	j	h	d	b	c	i	x
w	m	r	g	l	o	h	j	k	b	a	d	s	v	e	q	t	i	f	u	n	p	x	z	c	y
j	v	b	m	h	g	f	i	c	d	z	a	k	l	x	s	w	e	o	t	p	y	q	u	r	n
r	f	c	l	y	z	g	v	j	w	m	p	t	h	w	e	q	u	a	x	b	k	o	d	s	i
c	r	h	u	a	x	w	p	n	q	e	b	f	m	s	t	l	y	d	z	j	o	i	v	k	g
m	u	o	p	j	t	k	d	f	c	h	g	e	y	z	a	n	l	x	s	i	b	r	q	v	w

The resulting trace would be: Down, Left, Up, Right, Down, Left, Down.

Figure 9 A trace of the password "daisies"

Modified CrossPassword

We also explored a modified version of this system designed to increase usability. This system uses a 13x13 grid, instead of a 26x26 grid to make it easier for users to visually scan the grid.

In addition, we no longer generate a Latin Square. Instead, we first randomly distribute the user's password in an empty grid. We first randomly select either a row or a column from our 26 choices. We then place the first letter somewhere in that row or column. For this example, say we select the 3rd column to start with. We then place the "a" somewhere in this first column. We then place the second

letter "m" in the row in which we have placed the first letter. We continue this scheme until the password has been placed.

For example:

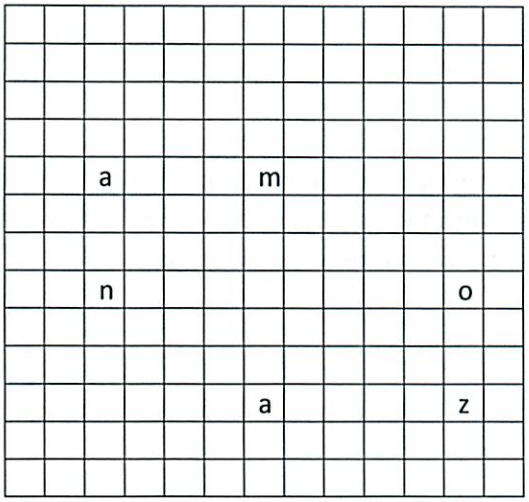


Figure 10 Password Characters Randomly Filled In

We then randomly fill in the remaining letters on the grid from the set of 26 lower case letters. We make sure each row and column only contains each letter only once by backtracking. For each spot we first start with the entire set [a...z]. We then remove all letters that are currently in the same row and column that we are in. We then randomly select a character from the remaining set.

This is not a Latin Square because we have a 13x13 Grid, but 26 possible characters.

With this system we must prevent an attacker from looking at a grid a certain amount of times. A user can only look at up to 4 random grids before their IP address is locked out. In addition, a particular user account can only have 4 grids shown before that account is locked out as well, in the event the attacker is using a distributed attack.

GrIDsure

As a comparison, we will evaluate our proposal against GrIDsure as described in The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes.[3] GrIDsure is also a cognitive authentication scheme in which the user attempts to prove to the server that it knows a secret without actually revealing the secret.

At registration, the user is presented with a 5x5 grid and selects a pattern of 4 cells. We call this the *password* for consistency with our other methods.

[3] From R. Jhawar, P. Inglesant, N. Courtois, and M. A. Sasse, "Make mine a quadruple: Strengthening the security of graphical one-time pin authentication," in Proc. NSS 2011, pp. 81–88
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=06059963

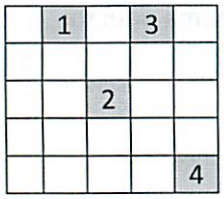


Figure 11 A length-4 pattern selected

At login, the user is again presented with a 5x5 grid, this time with a random digit in each cell. The user then transcribes the digits from his or her length-4 pattern into the nearby box. For consistency, we call this the *trace*. The digits differ each time because they are chosen randomly each time.

5	3	4	8	9
2	2	7	4	4
3	6	9	6	3
8	4	6	7	0
9	8	1	0	7

Figure 12 The password here would be "3987"

Analysis of Attacks

We evaluate the security of the three systems described above using a probability analysis. In eavesdropping on a user, an adversary may obtain the user's board/grid, the trace, or both. Given each of these pieces of information we see how the user may be compromised. The attacker may be able to just log in through brute force chance; or the attacker may be able to backsolve for the original password.

Original Off the Grid

The Off the Grid system offers users a unique password given the domain name. The user simply traces the domain name in the two phases described above.

Intercepted Grid (Theft or Video Observation)

The first attack may occur if an adversary obtains a user's grid. This is usually held on hand by the user, and may occur if their wallet is stolen.

If an attacker is able to retrieve the grid of a user, he has access to all of the user's passwords. The attacker can follow the Off the Grid protocol for any possible domain name in order to obtain the password for that site. This is especially true if the user sticks to the standard Off the Grid protocol. For example, the algorithm specifies starting in the first row.

However, the protocol author recommends making personal tweaks to the algorithm, for example, choosing a custom, personal, secret start location. If someone adopts a personal start location, this leads to 104 possible passwords if the attacker gets control of the grid. 104 choices is still easily brute forcible. Other tweaks can add further additional possibilities.

We do not, however, officially consider these tweaks in evaluating the algorithm. Nevertheless, the secrecy of the grid is paramount in preserving the integrity of the Off the Grid; if the grid is stolen, an adversary has access to all passwords for all websites.

Observing a Single Site (Internal Observation, SSL Middleman, or Phishing Site)

If an attacker instead obtains the user's password for a single site, only interactions with that website are compromised. In the description described above, the password "gaznegmacmzg" is obtained by following the two phases of the Off the Grid system for Amazon.com. If the attacker obtains "gaznegmacmzg," he can use it to log in the user's account for Amazon.

Rebuilding the Grid

With one password, the attacker obtains very little information regarding the grid, and the resulting passwords for other website domains. It is very difficult to use one, or even a collection of passwords, to rebuild the Grid. Given the stock Off the Grid implementation described, there are at least $\frac{(n!)^{2n}}{n^{n^2}}$ boards for an n-sized board, which leads to at least 9.337×10^{426} boards for $n = 26$. Even if you knew the start location (the first row), there are 26 possible slots for the first character. There are then 26 possibly for the next character, etc. In addition, because one wraps around, no information is leaked about boundaries. Even with many, many domain and password pairs it is infeasible to recreate the grid and/or generate new passwords for a given new domain name.

Brute Forcing a Password

Under the standard Off the Grid protocol (where we always use the first 6 characters of the domain name), there are 26^{12} possible submissions to the website. This is hard to brute force.

CrossPassword

Our CrossPassword implementation relies on the security of the Latin Square. Since each password must be made up of only lower case letters and no-repeating characters, the password can be searched using a 26×26 Latin Square as specified above, resulting in at least 9.337×10^{426} possible grids.

Intercepted Board

If the attacker gains access to the board the user sees along with the start location, the attacker gains very little information on the password of the user even after multiple board configurations are given. Because the first letter of the password will never be the start location, with many copies of the grid, the attacker could eventually see that there is one letter which is never the start location. This occurs 1/26 times, but an attacker needs ~50-70 boards to confidently say that the 1 letter will never show up.

With 78 boards they have a

$(\frac{1}{26})^3$ = .000005 chance of having a location never show by accident, so it is likely that the space that remains is the start location. This only reveals the first letter.

Brute Forcing Traces

An adversary can do a brute-force guess on the submission. We can use the fact that the direction changes each turn, thus Left and Right will always be followed by Up or Down along with the start location to realize that there are only 2 possible responses for each character of the password.

If the attacker doesn't know the length of the password, then they must try multiple combinations

$$\sum_{l=1}^{\infty} 2^l$$

This sum does not converge. However, if the attacker assumes a maximum length of 10, then there are 2046 possibilities. This is not very large.

$$\sum_{l=1}^{10} 2^l = 2046$$

Intercepting Just the Trace

If the attacker instead gains only access to the user's key inputs (but not the grid) through key-logging software, thus obtaining their input, they can never retrieve the password. The password will be impossible to obtain from only getting "up, right, up, left, down" and so on. The only information gained is the length of the password. Even with multiple traces, the password will be impossible to obtain without the accompanying board. However given the length of the password, in a brute force attack the user knows exactly how long the trace to input; an attacker can now do a smarter brute force attack of the exact length of the password.

If the attacker knows there are 6 characters in the password then there are only $2 \times 2 \times 2 \times 2 \times 2 \times 2 = 64$ possible combinations which we can then brute force. After finding out the length of a password through the trace, we know there are then 2^l possibilities given, l, the length of the password.

Lockout

Because this scheme is highly susceptible to brute force attacks with a $\frac{1}{2^l}$ probability of guessing correctly, we can reduce the effectiveness of brute force attacks by the frequency of the board change and lockout. If we lockout after every 4 attempts, then there is a chance of guessing a specific user's password of

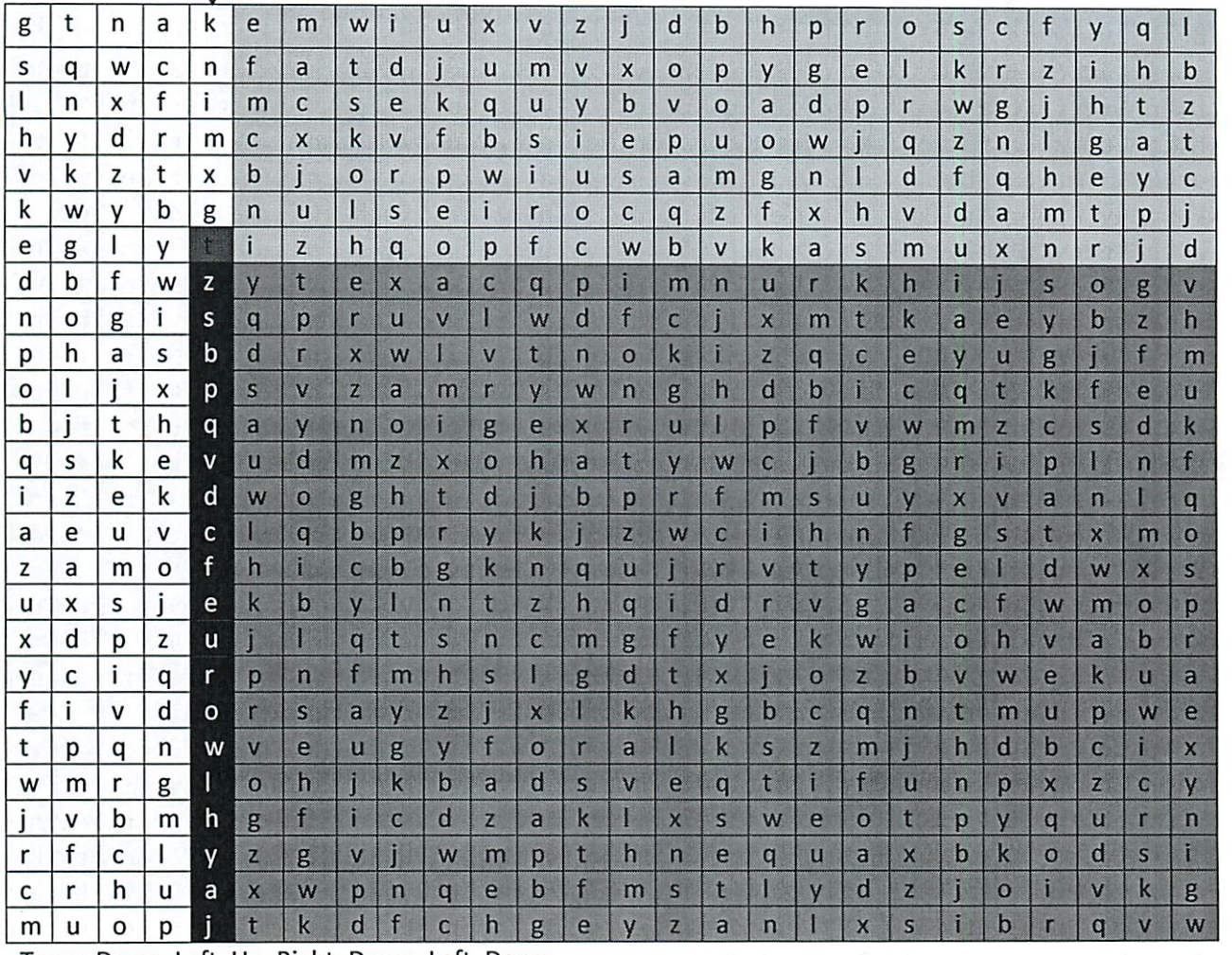
$$1 - \left(\frac{2045}{2046} * \frac{2044}{2045} * \frac{2045}{2046} * \frac{2044}{2045}\right) = .19\%$$

before being locked out given that all users choose passwords less than or equal to 10 characters. This means an attacker will likely figure out the password of about every 500th user. The attacker would need to use a range of IP addresses to avoid IP blocking.

Intercepted One Board and Trace: Password Recovery?

Lastly, in the case when an attacker gains information to the board, the start-location, and the trace. Even when the trace is intercepted along with its board, it is extremely difficult to recover the password with one of these sets as can be seen in the example below where the password is "daisies."

Start: Vertical



g	t	n	a	k	e	m	w	i	u	x	v	z	j	d	b	h	p	r	o	s	c	f	y	q	l
s	q	w	c	n	f	a	t	d	j	u	m	v	x	o	p	y	g	e	l	k	r	z	i	h	b
l	n	x	f	i	m	c	s	e	k	q	u	y	b	v	o	a	d	p	r	w	g	j	h	t	z
h	y	d	r	m	c	x	k	v	f	b	s	i	e	p	u	o	w	j	q	z	n	l	g	a	t
v	k	z	t	x	b	j	o	r	p	w	i	u	s	a	m	g	n	l	d	f	q	h	e	y	c
k	w	y	b	g	n	u	l	s	e	i	r	o	c	q	z	f	x	h	v	d	a	m	t	p	j
e	g	l	y	t	i	z	h	q	o	p	f	c	w	b	v	k	a	s	m	u	x	n	r	j	d
d	b	f	w	z	y	t	e	x	a	c	q	p	i	m	n	u	r	k	h	i	j	s	o	g	v
n	o	g	i	s	q	p	r	u	v	l	w	d	f	c	j	x	m	t	k	a	e	y	b	z	h
p	h	a	s	b	d	r	x	w	l	v	t	n	o	k	i	z	q	c	e	y	u	g	j	f	m
o	l	j	x	p	s	v	z	a	m	r	y	w	n	g	h	d	b	i	c	q	t	k	f	e	u
b	j	t	h	q	a	y	n	o	i	g	e	x	r	u	l	p	f	v	w	m	z	c	s	d	k
q	s	k	e	v	u	d	m	z	x	o	h	a	t	y	w	c	j	b	g	r	i	p	l	n	f
i	z	e	k	d	w	o	g	h	t	d	j	b	p	r	f	m	s	u	y	x	v	a	n	l	q
a	e	u	v	c	l	q	b	p	r	y	k	j	z	w	c	i	h	n	f	g	s	t	x	m	o
z	a	m	o	f	h	i	c	b	g	k	n	q	u	j	r	v	t	y	p	e	l	d	w	x	s
u	x	s	j	e	k	b	y	l	n	t	z	h	q	i	d	r	v	g	a	c	f	w	m	o	p
x	d	p	z	u	j	l	q	t	s	n	c	m	g	f	y	e	k	w	i	o	h	v	a	b	r
y	c	i	q	r	p	n	f	m	h	s	l	g	d	t	x	j	o	z	b	v	w	e	k	u	a
f	i	v	d	o	r	s	a	y	z	j	x	l	k	h	g	b	c	q	n	t	m	u	p	w	e
t	p	q	n	w	v	e	u	g	y	f	o	r	a	l	k	s	z	m	j	h	d	b	c	i	x
w	m	r	g	l	o	h	j	k	b	a	d	s	v	e	q	t	i	f	u	n	p	x	z	c	y
j	v	b	m	h	g	f	i	c	d	z	a	k	l	x	s	w	e	o	t	p	y	q	u	r	n
r	f	c	l	y	z	g	v	j	w	m	p	t	h	n	e	q	u	a	x	b	k	o	d	s	i
c	r	h	u	a	x	w	p	n	q	e	b	f	m	s	t	l	y	d	z	j	o	i	v	k	g
m	u	o	p	j	t	k	d	f	c	h	g	e	y	z	a	n	l	x	s	i	b	r	q	v	w

Trace: Down, Left, Up, Right, Down, Left, Down

Figure 13: A board, start location, and trace have been intercepted. It remains difficult to recover the password. Possibilities for the first letter are in black, 2nd letter in dark gray.

There are many possible combinations of letters that will satisfy the trace. An attacker will have to obtain all combinations that satisfy the trace, then try to determine which is the password. For example in **Figure 13: A board, start location, and trace have been intercepted. It remains difficult to recover the password.**Figure 13, there are 19 options for the first letter (in black) then 19*21 possible options for the second letter (in medium gray) because we know our start column is the 5th column. However, because this is a Latin Square, we know that within these 399 cells exist all 26 letters.

Start: Vertical

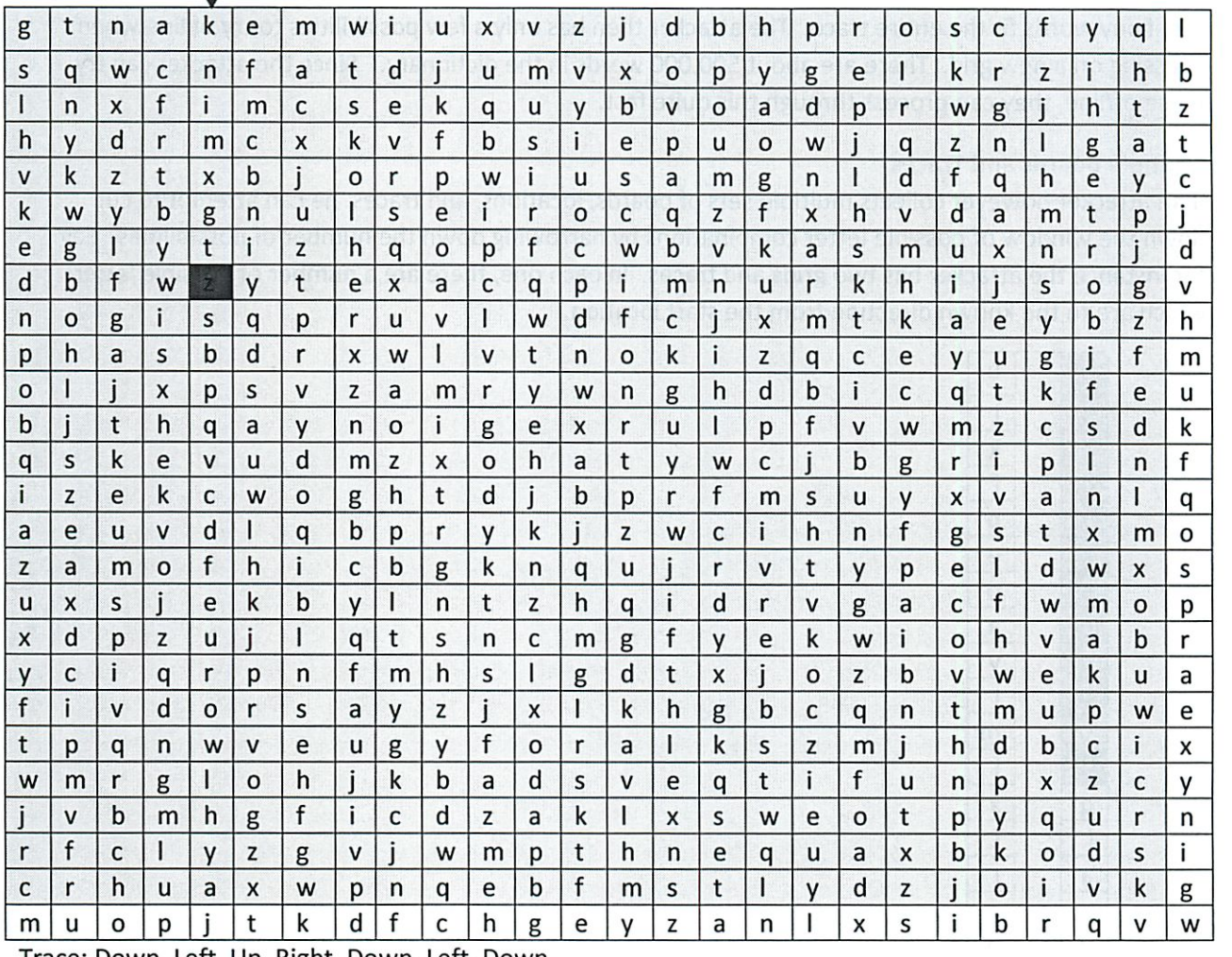
				↓																					
g	t	n	a	k	e	m	w	i	u	x	v	z	j	d	b	h	p	r	o	s	c	f	y	q	l
s	q	w	c	n	f	a	t	d	j	u	m	v	x	o	p	y	g	e	l	k	r	z	i	h	b
l	n	x	f	i	m	c	s	e	k	q	u	y	b	v	o	a	d	p	r	w	g	j	h	t	z
h	y	d	r	m	c	x	k	v	f	b	s	i	e	p	u	o	w	j	q	z	n	l	g	a	t
v	k	z	t	x	b	j	o	r	p	w	i	u	s	a	m	g	n	l	d	f	q	h	e	y	c
k	w	y	b	g	n	u	l	s	e	i	r	o	c	q	z	f	x	h	v	d	a	m	t	p	j
e	g	l	y	t	i	z	h	q	o	p	f	c	w	b	v	k	a	s	m	u	x	n	r	j	d
d	b	f	w	z	y	t	e	x	a	c	q	p	i	m	n	u	r	k	h	i	j	s	o	g	v
n	o	g	i	s	q	p	r	u	v	l	w	d	f	c	j	x	m	t	k	a	e	y	b	z	h
p	h	a	s	b	d	r	x	w	l	v	t	n	o	k	i	z	q	c	e	y	u	g	j	f	m
o	l	j	x	p	s	v	z	a	m	r	y	w	n	g	h	d	b	i	c	q	t	k	f	e	u
b	j	t	h	q	a	y	n	o	i	g	e	x	r	u	l	p	f	v	w	m	z	c	s	d	k
q	s	k	e	v	u	d	m	z	x	o	h	a	t	y	w	c	j	b	g	r	i	p	l	n	f
i	z	e	k	c	w	o	g	h	t	d	j	b	p	r	f	m	s	u	y	x	v	a	n	l	q
a	e	u	v	d	l	q	b	p	r	y	k	j	z	w	c	i	h	n	f	g	s	t	x	m	o
z	a	m	o	f	h	i	c	b	g	k	n	q	u	j	r	v	t	y	p	e	l	d	w	x	s
u	x	s	j	e	k	b	y	l	n	t	z	h	q	i	d	r	v	g	a	c	f	w	m	o	p
x	d	p	z	u	j	l	q	t	s	n	c	m	g	f	y	e	k	w	i	o	h	v	a	b	r
y	c	i	q	r	p	n	f	m	h	s	l	g	d	t	x	j	o	z	b	v	w	e	k	u	a
f	i	v	d	o	r	s	a	y	z	j	x	l	k	h	g	b	c	q	n	t	m	u	p	w	e
t	p	q	n	w	v	e	u	g	y	f	o	r	a	l	k	s	z	m	j	h	d	b	c	i	x
w	m	r	g	l	o	h	j	k	b	a	d	s	v	e	q	t	i	f	u	n	p	x	z	c	y
j	v	b	m	h	g	f	i	c	d	z	a	k	l	x	s	w	e	o	t	p	y	q	u	r	n
r	f	c	l	y	z	g	v	j	w	m	p	t	h	n	e	q	u	a	x	b	k	o	d	s	i
c	r	h	u	a	x	w	p	n	q	e	b	f	m	s	t	l	y	d	z	j	o	i	v	k	g
m	u	o	p	j	t	k	d	f	c	h	g	e	y	z	a	n	l	x	s	i	b	r	q	v	w

Trace: Down, Left, Up, Right, Down, Left, Down

Figure 14: A board, start location, and trace have been intercepted. It remains difficult to recover the password. Possibilities for the third character are in light gray.

For the third character, we know we are going up again. We know that we are somewhere within the light grey. Note that we know that we can't be on the bottom row for this character. This is 19*25 possibilities; we know again that again all 26 letters exist because it is a Latin Square.

Start: Vertical



g	t	n	a	k	e	m	w	i	u	x	v	z	j	d	b	h	p	r	o	s	c	f	y	q	l
s	q	w	c	n	f	a	t	d	j	u	m	v	x	o	p	y	g	e	l	k	r	z	i	h	b
l	n	x	f	i	m	c	s	e	k	q	u	y	b	v	o	a	d	p	r	w	g	j	h	t	z
h	y	d	r	m	c	x	k	v	f	b	s	i	e	p	u	o	w	j	q	z	n	l	g	a	t
v	k	z	t	x	b	j	o	r	p	w	i	u	s	a	m	g	n	l	d	f	q	h	e	y	c
k	w	y	b	g	n	u	l	s	e	i	r	o	c	q	z	f	x	h	v	d	a	m	t	p	j
e	g	l	y	t	i	z	h	q	o	p	f	c	w	b	v	k	a	s	m	u	x	n	r	j	d
d	b	f	w	z	y	t	e	x	a	c	q	p	i	m	n	u	r	k	h	i	j	s	o	g	v
n	o	g	i	s	q	p	r	u	v	l	w	d	f	c	j	x	m	t	k	a	e	y	b	z	h
p	h	a	s	b	d	r	x	w	l	v	t	n	o	k	i	z	q	c	e	y	u	g	j	f	m
o	l	j	x	p	s	v	z	a	m	r	y	w	n	g	h	d	b	i	c	q	t	k	f	e	u
b	j	t	h	q	a	y	n	o	i	g	e	x	r	u	l	p	f	v	w	m	z	c	s	d	k
q	s	k	e	v	u	d	m	z	x	o	h	a	t	y	w	c	j	b	g	r	i	p	l	n	f
i	z	e	k	c	w	o	g	h	t	d	j	b	p	r	f	m	s	u	y	x	v	a	n	l	q
a	e	u	v	d	l	q	b	p	r	y	k	j	z	w	c	i	h	n	f	g	s	t	x	m	o
z	a	m	o	f	h	i	c	b	g	k	n	q	u	j	r	v	t	y	p	e	l	d	w	x	s
u	x	s	j	e	k	b	y	l	n	t	z	h	q	i	d	r	v	g	a	c	f	w	m	o	p
x	d	p	z	u	j	l	q	t	s	n	c	m	g	f	y	e	k	w	i	o	h	v	a	b	r
y	c	i	q	r	p	n	f	m	h	s	l	g	d	t	x	j	o	z	b	v	w	e	k	u	a
f	i	v	d	o	r	s	a	y	z	j	x	l	k	h	g	b	c	q	n	t	m	u	p	w	e
t	p	q	n	w	v	e	u	g	y	f	o	r	a	l	k	s	z	m	j	h	d	b	c	i	x
w	m	r	g	l	o	h	j	k	b	a	d	s	v	e	q	t	i	f	u	n	p	x	z	c	y
j	v	b	m	h	g	f	i	c	d	z	a	k	l	x	s	w	e	o	t	p	y	q	u	r	n
r	f	c	l	y	z	g	v	j	w	m	p	t	h	n	e	q	u	a	x	b	k	o	d	s	i
c	r	h	u	a	x	w	p	n	q	e	b	f	m	s	t	l	y	d	z	j	o	i	v	k	g
m	u	o	p	j	t	k	d	f	c	h	g	e	y	z	a	n	l	x	s	i	b	r	q	v	w

Trace: Down, Left, Up, Right, Down, Left, Down

Figure 15: A board, start location, and trace have been intercepted. It remains difficult to recover the password. Possibilities for the fourth character are in light gray.

For the fourth character, we know that we can be anywhere except the last column (since we just went up) and the last row (since we are going right). There are 625 possible cells where we can be at now, again containing all 26 characters, by the properties of a Latin Square.

It continues like this for the rest of the password. There are

$$19 \times 26^6 = 25.8 \times 10^9$$

possible sequences of grids which we visited, leading to the same number of possible unique passwords (since it is a Latin Square). Thus for each board, location, and trace there are significantly too many possible passwords, that knowing the board and the trace does not reveal the underlying password.

Dictionary Word

If the user selects a dictionary word, it is much easier for the attacker to recover the user's original password. The attacker can attempt to run every dictionary word through the observed trace offline to see if any words fit the entire trace. The attacker then has only a few possibilities to try online when guessing on a new grid. There are about 500,000 words in the dictionary.[4] Since the attacker can try these offline, they can process through this quite fast.

Multiple Boards and Traces

If an attacker however collects multiple sets of boards, locations, and traces, he can attempt to cut down the window of possible letter combinations by narrowing down the number of possibilities. Say for instance the attacker has two grids and traces. In each one, there are a number of possible letters which are to the known direction from the start location.

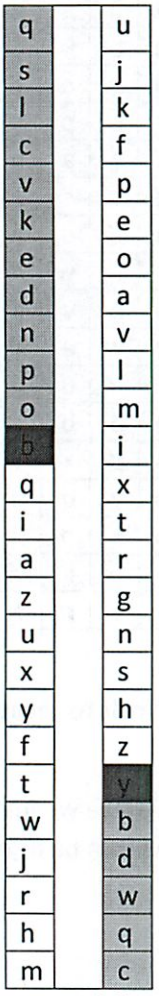


Figure 16: Two different first columns where the first letter is c.

The attacker will then look at the union of the two shaded sections. In this case they can see that the only characters in the union are c, d, and q. With yet another grid and trace, the attacker can narrow this.

[4] https://dazzlepod.com/uniqpass/

With 1 trace, the attacker will know have 12.5 possible characters on average; with 2 traces, there are 6.25 possible characters on average. With 5 traces there is likely to be only 1 possible character.

1 trace/board	12.5 possible characters
2 traces/boards	6.25 possible characters
3 traces/boards	3.125 possible characters
4 traces/boards	1.5625 possible characters
5 traces/boards	.78125 possible characters

Table 1. Expected number of letters that will be still be present at random after specified number of traces/boards

The attacker can then step through each letter of the password looking at one letter at a time; using the grids and traces he already has. He can thus recover the password with approximately 5-6 complete grids and traces.

Modified CrossPassword

While our CrossPassword implementation relies on the security of the Latin Square, the modified version is no longer a Latin Square having a reduced size, 13 × 13, but still using all 26 letters as possible letters.

Intercepted Board

Unlike our previous implementation, the adversary gains information looking at even a single board and start location, but without a trace. After looking at a few boards, the adversary may be able to determine all the letters of the password. To reiterate, the attacker just needs to refresh the log in screen!

The adversary is able to get a copy of the board and start location by simply visiting the login page of the server and entering the target's username. The fatal flaw is that the password letter is always present in the start row/column. The other letters are present with a 12/25 chance. To reiterate, we know that one letter will show up 100% of the time, with the remaining 12 letters having been randomly selected from the remaining 25 letters. This already cuts down the possibilities for the first letter of the underlying password to about ½ of all possible letters. By simply refreshing the page and getting a new board to see, the adversary can cut down the possible letters even more.

1 board	12 letters
2 boards	6
3 boards	3
4 boards	1.5
5 boards	.75

Table 2. Expected number of letters that will be present at random.

He continues this process until there is only 1 letter that is always present across all boards in the start row/column; this letter is the first letter of the password. This will take about 4-6 grids.

After the first letter is known, the adversary can repeat the process one letter at a time to calculate all letters in the password. The adversary can also reuse the grids they have already retrieved; there is no need to get new grids. The adversary knows to stop (i.e. the length of the password) when there is no longer a letter that is present 100% of the time in the next row/column.

Intercepted Trace

Similar to our Original CrossPassword implementation, the adversary gains very information about the underlying grid when they observe just a trace, or when they observe multiple traces, without the accompanying board. They do however know the length of the password, which greatly reduces the space they must brute force over.

Brute Forcing

The Modified CrossPassword is also vulnerable to the same brute force attack as the regular system.

$$\sum_{l=1}^{\infty} 2^l$$

Intercepted Board and Trace

Lastly, there is the possibility that the attacker has all information: the board, the start-location, and the trace. This equates to the information gained from have the board and its accompanying starting location, and the information gained from having the trace. The adversary can do a similar process using both the direction from the trace, as well as eliminating letters which only show up once. This leads to a greatly reduced number of possible letters.

Using just information for the trace:

1 trace/board	6 possible letters
2 traces/boards	3 letters
3 traces/boards	1.5 possible letters
4 traces/boards	.75 possible letters

Table 3. Expected number of letters that will be still be present at random after specified number of traces/boards

So with two traces/boards what is the probability that a letter is present twice in given direction at random?

$$(\frac{6}{26})^2 = 5\%$$

This means that is a letter is present, it is likely that it since it is the password, not that this letter has appeared at random. Thus with just two grids/traces you are likely to have the password.

GrIDsure

There are 25^4possible combinations of length-4 patterns. However, there are 10^4 possible inputs to return.

Intercepted Board

If an attacker is able to successfully intercept the board and no other information, he gains no information about the password. Each slot in the board is randomly filled with an integer. Even after intercepting multiple copies, the attacker gains no information for the user's password.

Brute Forcing

There are 10^4 possibilities, which is ok by our standards.

Intercepted Trace

Equally, if an attacker is able to intercept the 4-digit trace but not the board, again no information is gained about the user's password. Each number can be equally represented by any of the 25 slots.

Intercepted Board and Trace

We cannot exactly map one "trace" from the grid to the squares that the user selected because there are 25 grid locations but only 10 digits. Thus each digit will be in the grid an average of 2.5 times.

So if an attacker is able to intercept the board and the trace, after intercepting multiple copies, he would be able to determine the user's password. We can begin by looking at the first digit, because we know each digit is located on the grid an average of 2.5 times, we can say that the first digit is located in 3 locations on the grid, one of which will correspond to the first slot of the password. The next time we intercept a new board and trace, we will again get a new digit that is located in approximately 3 locations on the grid. We know for a fact that one of these locations must overlap because it will be a part of the user's password.

However, there is a small possibility that the other 2 locations may overlap with the original 2 other locations and thereby having more than one possibility for the underlying cell which is part of the user's password. The probability of the second two location overlapping exactly with the first two locations is $^2/_{24} \times {}^1/_{23} = {}^1/_{276}$. This is a low probability of gaining conflicting information about the slot that corresponds to the each "character" of the password. We need to extend this to multiple characters in the password. After intercepting more copies, this probability decreases significantly. So we know that at each new board, we will likely gain information to determine the slot that corresponds to the first slot in the password. This same analysis can be done to determine the slot for the second through fourth slot of the password.

In the original paper, the authors conduct a Monte Carlo simulation on 1,000,000 attacks and find that they need an average of 2.66 grids/traces, with a maximum of 8 grids/traces to recover the original password.[5]

# of grids/traces captures	# of grids reverse engineered (of 1,000,000)
1	129
2	422230
3	496400
4	72871
5	7527
6	770
7	69
8	4

Table 4. Number of captures needed to reverse engineer a password.

[5] From R. Jhawar, P. Inglesant, N. Courtois, and M. A. Sasse, "Make mine a quadruple: Strengthening the security of graphical one-time pin authentication," in Proc. NSS 2011, pp. 81–88 http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=06059963

Other Factors

We evaluate each system according to the criteria set out in The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes.[6]

Improvements to Criteria

Resilient-to-Physical Observations Category

We think that the **Resilient-to-Physical Observations** category should be split in two: **casual observation** and **video observation**. Casual observation is if an attacker is just able to watch the user enter their password once. This is feasible for short passwords and/or if the user types slow. An attacker can see which keys are hit on the keyboard. This is especially true if the user types slowly, has a short, and/or easily remember-able password.

However, the attacker seeing the user trace out the password on the grid once would have trouble remembering the entire grid, preventing the total loss of the password scheme. For that specific domain name, many attackers would have trouble remembering the sequence of 12 random characters, providing some additional security.

Video observation is defined as the attacker having the full ability to carefully watch and study users' movements because the attacker is able to pause and replay the user's log in actions.

Resilient-to-Throttled-Guessing

To better demonstrate the differences between our protocols, we assign **Resilient-to-Throttled-Guessing** if there are more than 10 possible choices all of equal weight. This is much small than the original paper requires. The paper considers $\frac{1}{10^4}$ choices to be NOT **Resilient-to-Throttled-Guessing**.

Inherently-Discoverable

Must the user seek out the new password system? Or does the server require that the user use it? Often new schemes that fit within the structure of existing passwords remain undiscoverable to the user. We want to highlight schemes where the website helps the user discover them. A scheme gets a YES here if the server is required to notify and teach the user of the new scheme.

Resilient-to-SSL-Proxy-Man-in-the-Middle

Assume that there is someone who is listening in on the wire who can decrypt SSL, for example, a corporate SSL proxy. Does this person have enough information to log in? YES, if they can do so after observing 1 log in. QUASI, if they must observe several log ins in order to have this power. We assume that the initial registration process is outside this scheme.

Allows -User-to-Choose-Any-Password

Does the system allow the user to choose any password (as defined by the usual set of characters allowed in a password)? Or does the system limit the user's set of password to a certain length or set of

---

[6] Bonneau, Joseph, Cormac Herley, Paul C. van Oorschot, Frank Stajanoy. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. University of Cambridge; Microsoft Research; Carleton University; University of Cambridge. Proc. IEEE Symp. on Security and Privacy 2012 ("Oakland 2012"). http://css.csail.mit.edu/6.858/2012/readings/passwords.pdf

characters? Or use a totally different memory scheme? Users might use the same password on multiple sites or have an external scheme to generate a password. One could argue that preventing a user from using the same password on each site is a good thing, but a password scheme should not do so by limiting the choice that a user has in selecting a password.

Denial-of-Serviceable

An active attacker can cause a denial-of-service attack by submitting a sufficient quantity of incorrect passwords such that the system locks the user out. Lockouts can add additional security by preventing more than a handful of guesses by the attacker. However, they can considerably impede usability as they can require a user to either wait or to seek out help from a system administrator. If these are tied to a user account, an attacker can deliberately use up these guesses to mount a Denial-of-Service attack on the user. If this is possible with a few incorrect submissions from any IP address, we assign a YES here.

Original Off the Grid

Usability benefits

1. **Memorywise-Effortless YES** There are no secrets to be remembered in the base case. The description mentions a more advanced case, where the user could start at a different location, but we are assuming the base case where the user automatically selects the same location.
2. **Scalable-for-Users YES** The user only needs one grid for all of their sites.
3. **Nothing-to-Carry NO** User must carry 1 sheet of paper
4. **Physically-Effortless NO** The user must trace out their password on paper
5. **Easy-to-Learn NO** Using the same rubric as the paper does, the scheme is quite complicated
6. **Efficient-to-Use NO** The scheme requires a fair amount of effort for each authentication.
7. **Infrequent-Errors NO** Tracing out the password on the grid twice is easy to mess up.
8. **Easy-Recovery-from-Loss KINDA** If the user lost their Grid, they must have another copy of their Grid, or the key used to generate that Grid. A user can always reset their passwords on each site. The paper rates generic passwords as **Easy-Recovery-from-Loss YES**.
9. **Inherently-Discoverable NO** A user must learn about this scheme by visiting the GRC website.
10. **Allows -User-to-Choose-Any-Password NO** The password is based off of the domain name of the site.

Deployability benefit

1. **Accessible NO** There could conceivably be a braille-based grid, but not at this moment. In addition, someone with poor motor control will find this scheme very difficult.
2. **Negligible-Cost-per-User YES** The user is required to print one sheet of paper which costs < 05 cents.
3. **Server-Compatibility YES** One of the primary benefits of this scheme is that it is compatible with existing servers which use passwords
4. **Browser-Compatibility YES** No special browser is needed
5. **Mature KINDA** The scheme has been published for some length of time; at least one Android app exists with support.
6. **Non-Proprietary YES** The scheme is published fully.

Security benefits

1. **Resilient-to-Physical Observations-Casual KINDA** The attacker would have to remember 12 random characters in order to observe the user's password for that site.
   With just a casual observation there is no way the attacker can memorize the entire Grid.
2. **Resilient-to-Physical Observations-Video NO** If the attacker can take a picture of the Grid, for example, a video camera over the shoulder, then the attacker would have access to all of the users' passwords assuming the user is using the standard Of the Grid scheme.
3. **Resilient-to-Targeted-Impersonation YES** Personal knowledge cannot help for the Off the Grid scheme. However, the normal password recovery mechanisms of the website remain, which are generally very vulnerable to Targeted Impersonation.
4. **Resilient-to-Throttled-Guessing YES** The user's password is 12 random alphanumeric characters. This means there are 26^{12} possible passwords.
5. **Resilient-to-Unthrottled-Guessing YES** There are 26^{12} possible passwords.
6. **Resilient-to-Internal-Observation NO** Off the Grid reduces to a normal 12 character password unique for each domain. This password is the same for each log in.
7. **Resilient-to-SSL-Proxy-Man-in-the-Middle NO** The password is the same for each log in; it must be protected with some additional protection (such as SSL) in transit.
8. **Resilient-to-Leaks-from-Other-Verifiers YES** Ideally the server should be hashing the password. Regardless, each domain has a unique password so leaking one password does not give one feasible information about another domains' password.
9. **Resilient-to-Phishing NO** If the attacker is able to spoof the domain name of the site, then the user will follow the same trace on the grid, providing the attacker their password.
10. **Resilient-to-Theft NO!** If the attacker gets your grid, it's game over, assuming you are sticking to the base Off the Grid algorithm. The author suggests that you make small personal tweaks to the algorithm in order to add resilience to theft.
11. **No-Trusted-Third-Party YES** The third party provides the code to generate the grid. However, that code runs in JavaScript on your local computer, allowing you to verify that the code is actually generating a unique grid and is not sending a copy to the third party. One could also write ones' own implementation of the Grid generation scheme to be sure.
12. **Requires-Explicit-Consent YES** The user must trace their password on the grid and then enter it onto the computer.
13. **Unlinkable YES** Since each user's Grid is so different, there is no feasible way to link users using the same scheme.
14. **Denial-of-Service-able NO** This is the same as normal passwords. Under a normal password system, services generally do not add a lockout provision.

CrossPassword

Goal: prevent from seeing over wire

Note all are for the actual log in experience. This analysis does not consider creating a password; the process of which is similar to traditional password schemes.

Usability benefits

1. **Memorywise-Effortless NO** The user must remember a password to use CrossPassword. Ideally, that password should be different between sites. Since we only allow lowercase alphabetic

characters without repeating letters, we may prevent users from using the same password on a site running CrossPassword than the user uses on all of their sites.

2. **Scalable-for-Users NO** Ideally the user has a different password for each site
3. **Nothing-to-Carry YES** There is nothing to carry
4. **Physically-Effortless NO** The user must trace out their password on-screen
5. **Easy-to-Learn NO** Using the same rubric as the paper, the scheme is quite complicated
6. **Efficient-to-Use NO** The scheme requires a fair amount of effort for each authentication.
7. **Infrequent-Errors NO** Tracing out the password on screen is easy to mess up
8. **Easy-Recovery-from-Loss YES** CrossPassword falls back on the same recovery mechanisms as traditional password sites, which is rated YES in the paper.
9. **Inherently-Discoverable YES** A user will discover the CrossPassword scheme when attempting to create an account on a server that uses CrossPassword
10. **Allows -User-to-Choose-Any-Password NO** The user can only choose a password using the letters [a...z] and the user cannot repeat the same characters twice, as in "aardvark."

Deployability benefit

1. **Accessible NO** A screen reader would be tedious to use. In addition, someone with poor motor control will find this scheme very difficult.
2. **Negligible-Cost-per-User YES** There is no cost.
3. **Server-Compatibility NO** The server must be provisioned with a new authentication library.
4. **Browser-Compatibility YES** No special browser is needed
5. **Mature NO** We are proposing it here
6. **Non-Proprietary YES** The scheme is published fully.

Security benefits

1. **Resilient-to-Physical Observations-Casual POSSIBLY** If the attacker could see the screen and the keyboard they could not uncover the user's password, unless the user traces the password with their finger.
2. **Resilient-to-Physical Observations-Video POSSIBLY** Even with being able to study the user as they enter their password, the attacker would not be able to recover a user's password, unless the user traces the password with their finger. This is one of the major design goals of this system.
3. **Resilient-to-Targeted-Impersonation YES** Personal knowledge cannot help for the Off the Grid scheme. However, the normal password recovery mechanisms of the website remain, which are generally very vulnerable to Targeted Impersonation.
4. **Resilient-to-Throttled-Guessing YES** An attacker can only submit two tracers per grid/start location. After two tries, the server will issue a new grid. The user then gets two more tries at a trace submission before the account is locked until an email loop is performed.
5. **Resilient-to-Unthrottled-Guessing NO** Due to the very small number of possible responses (for example, 2^6=64 for a 6 character password, there are very few bits of entropy so the system falls fast.
6. **Resilient-to-Internal-Observation QUASI** This is the major design goal of this system. An attacker needs 5-6 observations of the *grid*, *Start Location*, and *trace* in order to crack the password. This is sharply reduced if the user picks a dictionary word, however.

7. **Resilient-to-SSL-Proxy-Man-in-the-Middle QUASI** This is the same as Internal-Observations. If a listener on the wire who was able to remove the SSL encryption, then they would need 5-6 observation in order to recover the password.
8. **Resilient-to-Leaks-from-Other-Verifiers NO** The password is stored in plain text on the server in order for the server to verify the password. This is not good practice.
9. **Resilient-to-Phishing YES** An attacker with just one trace could not submit that trace to another server, because the grid is randomized each time. At attacker could mount a man-in-the-middle attack and proxy the grid, but the rubric in the paper does not penalize for this.
10. **Resilient-to-Theft YES** There is nothing to steal
11. **No-Trusted-Third-Party YES** There are no 3rd parties involved
12. **Requires-Explicit-Consent YES** The user must trace their password on the computer and enter the trace.
13. **Unlinkable YES** Like passwords, this scheme is unlinkable.
14. **Denial-of-Service-able YES** An attacker can lock out an account by trying an incorrect password 4 times.

Modified CrossPassword

The modified CrossPassword is more **Efficient-to-Use** and has less errors (**Infrequent-Errors**), however at the cost of a slightly decreased **Resilient-to-Physical Observations-Casual** and **Resilient-to-Physical Observations-Video** if a user traces the grid because of the smaller grid. The degree is reduced, but the broad scores remain the same.

However, **Resilient-to-Internal Observation** and **Resilient-to-SSL-Proxy-Man-in-the-Middle** take big hits as an attacker can discover a user's password (or at least have a very high chance of finding it) using 5-10 copies of the random grid and start location. They don't need any copies of the trace, though having at least 1 would let them need less copies of the grid. This makes the scheme vastly weaker. **Resilient-to-Throttled-Guessing, Resilient-to-Unthrottled-Guessing** switch to no because the attacker has that high chance of recovering the password and might only need to make 1-2 guesses. In fact, we don't even have a category for how bad this is: **Crackable-From-Reloading-Log-In-Page**?

Because we now need to protect from the attacker seeing the grid multiple times, modified CrossPassword is even more **Denial-of-Service-able**.

GrIDsure

GrIDsure is evaluated in The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. Here we evaluate the new metrics we have introduced and make additional comments about some metrics.

It is important to note that the authors rated it as not **Resilient-to-Throttled-Guessing** or **Resilient-to-Unthrottled-Guessing** because the space of possible is so small (10^4). In this paper, we rated our other schemes as **Resilient-to-Throttled-Guessing** if the attacker has less than 10 possible choices, so we would rate this as **Resilient-to-Throttled-Guessing** if it has a rate limiter/lockout.

Because the server must tell the user about the scheme, GrIDsure is **Inherently-Discoverable**. However, it requires users to remember a sequence of 4 unmarked boxes in a 5x5 grid. Thus it is clearly not **Allows -User-to-Choose-Any-Password**.

When the user does not place their finger to the screen, it is **Resilient-to-Physical Observations-Casual.**

Because of the small number of possibilities it is not **Resilient-to-Internal-Observation**, **Resilient-to-SSL-Proxy-Man-in-the-Middle**, or **Resilient-to-Physical Observations-Video**. With two observations, it is pretty much game over, as the attacker is able to discover the original sequence of boxes.

Comparison Table

	Of the Grid	CrossPass word	Modified CrossPass word	GrIDsure
Memorywise-effortless	Yes	No	No	No
Scalable-for-users	Yes	No	No	No
Nothing-to-carry	No	Yes	Yes	Yes
Physically-effortless	No	No	No	No
Easy-to-Learn	No	No	No	Yes
Efficient-to-Use	No	No	No (More)	Quasi
Infrequent-Errors	No	No	No (More)	Quasi
Easy-Recovery-from-Loss	Kinda	Yes	Yes	Yes
Inherently-Discoverable	No	Yes	Yes	Yes
Allows -User-to-Choose-Any-Password	No	No	No	No
Accessible	No	No	No	No
Negligible-Cost-per-User	Yes	Yes	Yes	Yes
Server-Compatibility	Yes	No	No	No
Browser-Compatibility	Yes	Yes	Yes	Yes
Mature	Kinda	No	No	No
Non-Proprietary	Yes	Yes	Yes	No
Resilient-to-Physical Observations-Casual	Kinda	Possibly	Possibly (Less)	Yes
Resilient-to-Physical Observations-Video	No	Possibly	Possibly (Less)	No
Resilient-to-Targeted-Impersonation	Yes	Yes	Yes	No
Resilient-to-Throttled-Guessing	Yes	Yes	No!	Yes
Resilient-to-Unthrottled-Guessing	Yes	No!	No (Less)	No
Resilient-to-Internal-Observation	No	~5-6	4-6 just log in screen!	~2
Resilient-to-SSL-Proxy-	No	Yes	No!	No

Man-in-the-Middle				
Crackable-From-Reloading-Log-In-Page	No	No	Yes	No
Resilient-to-Leaks-from-Other-Verifiers	Yes	No!	No!	No
Resilient-to-Phishing	No	Yes	Yes	No
Resilient-to-Theft	No!	Yes	Yes	Yes
No-Trusted-Third-Party	Yes	Yes	Yes	Yes
Requires-Explicit-Consent	Yes	Yes	Yes	Yes
Unlinkable	Yes	Yes	Yes	Yes
Denial-of-Service-able	No	Yes	Yes (More)	No

Table 5.: A Comparison of Off the Grid, CrossPassword, Modified CrossPassword, and GrIDsure.

Usability

We will now explore what the field of usability tells us about our password schemes. [7] The three core tenants of usability are: learnability, efficiency, and safety.

At the core, the simpler a system is, the more it will be used. Security is often a tradeoff between usability and security. A successful scheme should add security, without impacting usability too much.

Learnability

Discoverability

In order for a system to start being used, it must be discoverable.

CrossPassword is more discoverable than Off the Grid because the website you are creating an account with can let you know that the website uses CrossPassword. It is inherently discoverable. Off the Grid requires that you hear about the system in some way. Websites can still advise you of the presence of Off the Grid, but the Off the Grid system, as currently designed and designated, is not inherently discoverable.

Training

It's important that a user know how to use a particular system.

CrossPassword can be taught to users when they pick their password for the site. For example, sites could show users a video of how to use CrossPassword. Sites could also provide an interactive training tool using CrossPassword that uses JavaScript and HTML 5 to show the user how to trace their actual password. (Using the actual password would reveal the user's password to a shoulder surfing attacker, but this may be appropriate for a secure room. The password would be stored in the DOM during registration, but this happens with a normal registration system as well)

[7] Material from MIT's 6.813 User Interface classes by Prof. Rob Miller Spring 2012.

Mental Model

When users interact with a system, they form a mental model of how that system operates "behind the scenes."

We believe that once CrossPassword is explained to a user, it is easy for that user to form a mental model of the system. The server asks you to solve a puzzle and you solve it. In addition, the rationale behind the system is also clear; it is clear that this prevents you from sending your password over the wire for subsequent log ins. Users should be able to understand how the system works. Each log in is consistent with the rules of the system and ones' mental model of the system.

Efficiency

Each log in should not take a long time. This is because user's time is valuable. In addition, users will be more likely to keep using the system if it is fast.

Whereas Off the Grid requires users to trace the grid twice, CrossPassword only requires a user to trace the grid once.

Off the Grid also requires one to enter two characters for each letter in the domain name during its Phase 2. CrossPassword is more natural to use than Off the Grid because one can trace the system on the screen as one enters the keyboard traces. We feel that expert users of CrossPassword could use the arrow keys without taking their eyes off the screen. This could make password entry quite fast.

However, both CrossPassword and Off the Grid are slower than traditional password schemes, or even password managers, such as LastPass. Users may not want to adopt a system that is slower than what they already have.

Chunking

Research has shown that people can remember 7 ±2 pieces of information at once.[8] A piece of information could be one letter. When letters are combined into an English word, that word is now one piece of information. To reiterate: a collection of 7 random letters are 7 pieces of information. However, a word comprised of 7 letters is only 1 piece of information.

We can use this to evaluate whither a causal visual observer (shoulder surfer) could observe a password off the screen. With Off the Grid, it would be difficult for an attacker to remember 12 characters using just their short term memory. This is why we rated it as KINDA for Resilient-to-Physical Observations-Casual.

Fitts's Law

Fitts's Law is an estimate of the time it takes someone to point to an object or steer among objects.[9] The rule as formulated by Scott MacKenzie is as follows:[10]

---

[8] De Groot, A. D., *Thought and choice in chess*, 1965

[9] Paul M. Fitts (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, volume 47, number 6, June 1954, pp. 381–391.

[10] I. Scott MacKenzie and William A. S. Buxton (1992). Extending Fitts' law to two-dimensional tasks. Proceedings of ACM CHI 1992 Conference on Human Factors in Computing Systems, pp. 219–226. http://doi.acm.org/10.1145/142750.142794

$$T = a + b \log_2(1 + \frac{D}{W})$$

where:

- T is the average time taken to complete the movement
- a represents reaction time to start moving
- b stands for the speed of movement
- D is the distance from the starting point to the center of the target.
- W is the width of the target measured along the axis of motion. W can also be thought of as the allowed error tolerance in the final position, since the final point of the motion must fall within $\pm W/2$ of the target's center.

We can use a more specific form to study steering tasks, the time to move your hand through a tunnel of length D and width S:

$$T = a + b\,(\frac{D}{S})$$

The index of difficulty is now linear.

We can use this to measure the amount of time it takes someone to trace through the grid, if they trace the grid with their finger or mouse. Ideally the user should not do that to maintain **Resilient-to-Physical Observations-Casual** and **Resilient-to-Physical Observations-Video**.

Assume a = 0 and b=200ms/bit for a mouse, using the upper limit of the empirical study.[11] Assume the user must travel 26 cm to reach a 1 cm square block. If a user had to steer within a row, this leads to an approximate time to trace of

$$0 + .2(26) = 5.2 \text{ seconds for traveling a row or column.}$$

This is the worst case possibility: the user (worst case empirical user) is using a mouse to travel the entire length of a row/column and they cannot leave the row/column with their mouse at all. This is

Improving Usability

We can do things to improve usability. For example, we can shade every other row or column, alternating between row and column on every user input.

[11] Soukoreff, R. William, and I. Scott MacKenzie. Towards a standard for pointing device evaluation, perspectives on 27 years of Fitts' law research in HCI. York University. Department of Computer Science and Engineering. November 4, 2004. http://www.yorku.ca/mack/ijhcs2004.pdf

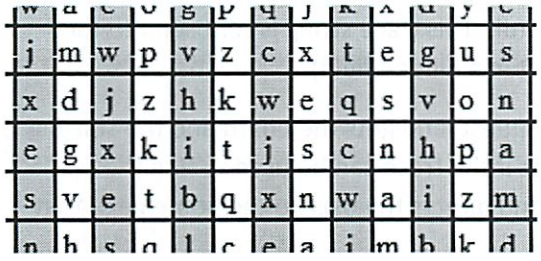


Figure 17 Every other column is shaded

This gives us two benefits. First, the user can now easily see if they should move horizontally or vertically next. Second, it is easier for the user to keep their eye in the same column/row as they scan the grid vertically/horizontally for their next letter. This should decrease mistakes as well as decrease the time it takes people to solve the grid.

Auto-Solver

It is possible to build a browser-based auto-solver for CrossPassword grids. This software would know the user's password and use that to automatically solve grid challenges. This would break **Resilient-to-Internal Observation** because the user's system would now need the password stored. However the system would still meet **Resilient-to-SSL-Proxy-Man-in-the-Middle** . It would do a great deal for usability, flipping **Physically-Effortless**, **Easy-to-Learn**, **Efficient-to-Use**, and **Infrequent-Errors** all to yes. In addition **Accessibility** would greatly improve.

Code

We built a prototype of CrossPassword to demonstrate the feasibility of implementing the system. Our code consists of a server-side component and a client-side component. The server runs a Flask Python server and keeps track of the user accounts in a PostgreSQL database. The client code is a Javascript file and HTML login page that is delivered by the server.

When the user requests to login as a specific username, the client-side code sends a request to the server for a board for that username. The server, upon receiving this request, generates a random 26x26 Latin square consisting of the lowercase letters of the alphabet and the starting coordinates and direction. This information is stored for the user in the database. The server then sends the board and starting square information back to the client. The client-side Javascript code inserts the board into the login page and marks the starting square and the starting direction (indicated by highlighting every other row/column and by text).

The user then traces out the password starting from the start square by using the arrow keys. The direction alternates between horizontal and vertical. For example, if the starting direction is horizontal, then the user will press either the left or right arrow key depending on where the first letter of the password is in that row. Then the board will change and indicate that the next direction is vertical. The user will then press either the up or down arrow key depending on where the second letter of the password is in that column. The user repeats this until the last letter of the password is traced, and then the user clicks the login button to submit the trace. As the user traced out the password, the client-side code converted the arrow key presses to the letters: 'u', 'd', 'l', 'r' and appended them to the end of the

trace string. The final trace string might be something like 'dlurdl'. This trace string is sent send to the server through a form POST request (the username is also sent in a hidden field).

To verify the trace, the server looks up the username in the request, and gets the board and the starting information. The server then looks at the trace string and tests if the trace can be used to find the password in the board. If so, then the server logs the user in and directs the user to the home page that is only accessible to signed-in users.

New users can be created through the registration page by providing a username and password (and confirmation password by entering the password again). The password is checked to make sure that it can be place into a board. In order to be place in a board and satisfy the Latin square constraints, the password must consist only of lowercase alphabet letters and must not have consecutive repeating letters. There is also a minimum password length requirement.

To see a working version of the implementation, go to crosspassword.herokuapp.com

Limitations of Current Code

We have not implemented any sort of lockout system in our sample implementation. This means that brute force attacks will be very easy to execute.

Conclusion

CrossPassword is not recommended as a password system. Modified CrossPassword turned out to be even weaker than we first imagined.

We tried to build a zero-knowledge interactive proof. A zero-knowledge interactive proof is one in which the prover needs to show that they know the solution to the verifier. The prover in this case is the user, and the verifier is the server. The verifier asks questions to the prover, who responds with an answer. If the prover does in fact know the answer, he or she will always answer the verifier's question correctly. If however, the prover does not actually know the answer, the prover may still get the question correct. However, over many questions the prover is likely to guess incorrectly at some point. Thus after enough guesses it is very unlikely that the prover is faking it. However, we only pose one question on each log in, which is not sufficient for a zero-knowledge interactive proof. Even with a super-aggressive lock out, CrossPassword still has false negative rates are well above the standards for cryptographic algorithms.

We controlled for the wrong thing. The password had a lot of information. However, the trace which we return to the server has very little information. For example, say we take a password and XOR the characters together to get 1 bit which is either yes or no. We transmit very little meaningful information of the password, but that very fact makes it easy for the attacker to guess!

The **Shannon entropy** of CrossPassword is l where l is the number of characters in the password. For example, a 6 character password has 6 bits of entropy. This makes it easy to brute force. In comparison, a single letter a-z has 26 possibilities or $log_2(26)$ = 4.7 bits *per letter*. Thus our 6 character password is almost the equivalent of a password of a single letter! If we allow upper and lower case,

digits, and 10 special characters, we have $log_2(72)$ = 6.2 bits of entropy, which is more than we currently have!

GrIDsure also has a reduction of entropy from 25^4 choices to 10^4 choices. However, 10^4 represents $log_2(10^4)$ = 13.3 bits of entropy, which is a good deal more than CrossPassword. Remember each additional bit doubles the number of possible passwords, and thus doubles the brute-force password search time.

However, GrIDsure is even unable to fulfill its design goal if an attacker has even two complete observations of grids and the corresponding PIN code. Figuring out the actual password is easier with GrIDsure, but it is harder to brute force. This may be a better tradeoff.

The Modified Scheme ends up being even worse because we reveal information about the password to the user. This ends up being disastrous because an attacker only needs to receive 4-6 copies of the grid from the server, and no data from the user actually entering their password!

On top of all this, our scheme is slower to enter than a traditional password, especially when used with a password manager.

This shows the inherent complexity in producing password schemes. There are many different objectives to try to achieve at once. Trading off some objectives produces different outcomes in security. Objectives cannot be traded off one-for-one, since the factors are not evenly weighted. There are many different possible attacks on a password scheme. It is difficult to keep all of the possible attacks in mind as one designs a particular scheme. Although this scheme was weak, it was interesting to evaluate exactly why it was weak and to think of possible attacks against the scheme.