Pico: No more passwords![*]

Frank Stajano

University of Cambridge Computer Laboratory

Abstract. From a usability viewpoint, passwords and PINs have reached the end of their useful life. Even though they are convenient for implementers, for users they are increasingly unmanageable. The demands placed on users (passwords that are unguessable, all different, regularly changed and never written down) are no longer reasonable now that each person has to manage dozens of passwords. Yet we can't abandon passwords until we come up with an alternative method of user authentication that is both usable and secure.
We present an alternative design based on a hardware token called Pico that relieves the user from having to remember passwords and PINs. Unlike most alternatives, Pico doesn't merely address the case of *web* passwords: it also applies to all the other contexts in which users must at present remember passwords, passphrases and PINs. Besides relieving the user from memorization efforts, the Pico solution scales to thousands of credentials, provides "continuous authentication" and is resistant to brute force guessing, dictionary attacks, phishing and keylogging.

1 Why users are right to be fed up

Remembering an unguessable and un-brute-force-able password was a manageable task twenty or thirty years ago, when each of us had to use only one or two. Since then, though, two trends in computing have made this endeavour much harder. First, computing power has grown by several orders of magnitude: once upon a time, eight characters were considered safe from brute force[1]; nowadays, passwords that are truly safe from brute force and from advanced guessing attacks[2] typically exceed the ability of ordinary users to remember them[3] [4]. Second, and most important, the number of computer-based services with which

[*] It's OK to skip all these gazillions of footnotes.

[1] The traditional DES-based `crypt(3)` didn't even *allow* a longer password.

[2] Thus, respectively, `long&Full^0f_$ymbo£$`, or even meaningless: `u4Hs9D6GdCVi`.

[3] Bruce Schneier, 1999: "Password crackers can now break anything that you can reasonably expect a user to memorize". `http://www.schneier.com/crypto-gram-9910.html#KeyLengthandSecurity`.

[4] It is in theory possible to mitigate the brute-forcing threat of ever-increasing attacker power by regularly increasing the number of rounds of hashing applied to the password before using the verification value. For other uses of passwords, such as file encryption, a similar countermeasure involves the use of a highly iterated key derivation function. Online systems could rate-limit the password guessing attempts, regardless of attacker power, as appropriately argued by Florêncio et al [12]. But ex-

In *Proc. Security Protocols Workshop 2011*, Springer LNCS.
Author's preprint, revision 61 of 2011-08-31 19:55:55 +0100 (Wed, 31 Aug 2011).

we interact has grown steadily, while our biological memory capacity has not increased: we no longer have just a single login password to remember but those for several computers, email accounts, dozens of web sites, encrypted files and various other services, not to mention a variety of PINs for our bank cards, phones, burglar alarms and so on.

Let's henceforth refer to all such password-requesting applications and services as "apps"[5]. For the implementer of a new app (say a new web site), passwords are the easiest and cheapest way of authenticating the user. And, since the password method is universally understood and used everywhere else, there is no apparent reason not to continue to use it[6].

This approach doesn't scale to the tens of passwords, passphrases and PINs each of us must use today. Users frequently complain that they can't stand passwords any longer. And they are right: the requests of computer security people have indeed become unreasonable, as highlighted by Adams and Sasse back in 1999 [1]. Users are told never to write down their passwords but they know they will face substantial hassle or embarrassment, if not total loss of their data or other assets, if they ever forget them: availability is often worth much more to them than confidentiality [3]. If they comply with the request not to write down their passwords, they'll want to choose something memorable. But, they're told, it must also be hard to guess[7]. Once they find a strong, memorable yet hard to guess password, the annoying computer security people also dictate that they can't reuse it anywhere else. It's hard to find several dozen different passwords that are all memorable and yet not guessable by someone who knows a little about the user. To add insult to injury, the next security request is that the passwords also be *changed* on a regular basis, to reduce exposure in case of compromise. Fat chance of ever remembering them all, then! Not to mention the difficulty of typing a complex password correctly, or of fixing a minor typo in it, without visual feedback ("***************").

Users are more than justified in being fed up. If the rules imposed by the computer security people are mutually incompatible, users cannot possibly follow them all, even when they are genuinely cooperative and willing to comply[8].

perience (cfr. Sklyarov's brilliant presentation at Passwords^11) shows that many widely deployed systems are still open to brute-force password cracking attacks that get more powerful with every new generation of CPUs—and now GPUs. We agree with Florêncio and Herley [11] that, for online systems, requiring users to adopt stronger passwords is an arrogant and counterproductive excuse for not adequately protecting the hashed password file.

[5] There is no connection between our "apps" and smartphone applications.

[6] As also observed by Bonneau and Preibusch [4], this situation is what economists call a *tragedy of the commons*: "What harm could my site cause by requesting a password, since everyone else's also does?" And yet, since everyone does, the burden collectively placed on users becomes unmanageable.

[7] Despite the easily retrieved digital footprints we all leave behind online.

[8] According to the insightful model of Beautement et al [3], users have a finite "compliance budget"—there's only so much hassle they'll take from the rule-makers before they'll stop complying. What we are saying here is that our password rules are con-

Imagine we could afford to start again from scratch, without concerns for backwards compatibility, and that our primary directive were to invent a user authentication method that no longer relied on users having to remember unguessable secrets. At a minimum, the new method would have to do better than passwords under at least three respects (refer to Table 1 on page 6 for a concise summary):

Memoryless: Users should not have to memorize any secrets[9].

Scalable: The effectiveness of the method should not be degraded even if the user had to authenticate to dozens or even thousands of different apps.

Secure: The alternative should be at least secure as passwords[10].

The design proposed in this paper meets these requirements. It is based on a token that remembers the user's authentication credentials. Any token-based alternative to passwords, however, should also address two additional concerns:

Loss-Resistant: If the token is lost or destroyed, the user must be able to regain access to the services.

Theft-Resistant: If the token is stolen, the thief must not be able to impersonate the user—even assuming that the thief can tamper with the token and access its insides.

The proposed design also meets these additional requirements. Its main disadvantage, aside from the obvious fact of having to carry a token[11], is that it requires changes to the verifying apps, which will be a barrier to deployment. For this reason, after describing the clean-slate design, in section 5 we evaluate possible variations aimed at trading off some security in order to increase convenience, backwards compatibility or market acceptance.

---

tradictory and impossible to follow all at once, even for fully cooperating users who have not yet exhausted their compliance budget. Even the people who *issue* those rules cannot possibly come up with fifty different strong passwords, never write them down anywhere, change them every month and remember them and type them out without errors whenever needed. These rules are mutually incompatible. Some of them *will* be broken. Which ones? Maybe the passwords will be reused. Maybe they will be written down. Maybe they will be changed and then changed back. By imposing an unsatisfiable set of rules, you can't anticipate *which* rules will be violated. But some of them definitely will.

[9] Some will argue that this requirement is not strictly a necessity and that it would be acceptable, for example, to rely on a single master passphrase. I shall call this inferior alternative **Quasi-Memoryless**. But the original motivation for my work on Pico is specifically to relieve users from having to remember *any* secrets.

[10] We actually mean: as secure as passwords would be *if* users were able to follow all those implausible and contradictory rules. Too easy otherwise.

[11] As Laurie and Singer [22] observe for their Neb: "users already carry [security tokens such as] keys and credit cards; the open question is how to persuade them that to view [this gadget] in the same class as the things they already carry".

2 Pico: a usable and secure memory prosthesis

This section gives an overview of the proposed system, Pico, and of all the benefits it claims to provide, besides the ones listed as mandatory in the previous section. Once again, a concise reference summary is in Table 1 on page 6. Benefits are typeset in boldface when first introduced. Refer to section 6 for related work.

The user has a trustworthy device called a Pico[12] that acts as a memory prosthesis and takes on the burden of remembering authentication credentials, transforming them from "something you know" to "something you have".

This shift addresses the first and fundamental requirement (MEMORYLESS) of no longer having to remember passwords. We could have trivially addressed it simply by writing the passwords down on a piece of paper; the solution, however, would fail the THEFT-RESISTANT requirement. Storing the passwords in encrypted form in a "password wallet" on a PDA or smart phone comes one step closer to the idea of the Pico. In section 4.1 we explain how the Pico offers the THEFT-RESISTANT benefit thanks to its Picosiblings.

The first few additional benefits are usability-related: let's review them.

A number of ingenious systems have been proposed to solve the problems of *web* passwords, from Single Sign-On (SSO, surveyed by Pashalidis and Mitchell [29]) to in-browser or in-cloud password wallets. However, even if they worked perfectly, solving the problem only for web passwords would still not satisfy users: if passwords and PINs are a nuisance, they are a nuisance everywhere, not just on the web. For the user, a password is a password, and it's annoying whether on the web or elsewhere. We must provide a comprehensive solution: **WORKS-FOR-ALL**. Pico replaces all passwords, not just web ones, including screen saver passwords, passphrases to unlock files on your local computer and the PINs of standalone devices such as your car stereo, burglar alarm, phone or smart cards.

In-browser and in-OS password wallets alleviate many burdens but are only available from the local host on which the credentials are kept. Pico, instead, like SSO and in-cloud wallets, works **FROM-ANYWHERE**.

Even if a system remembers passwords on behalf of the user, like a password wallet does, if there are many of them (and we promised with the SCALABLE benefit that there could be thousands) then the user faces the secondary problem[13] of selecting the appropriate credential for the circumstance. Pico's **NO-SEARCH** benefit means that the appropriate credentials are selected automatically when the user wishes to interact with an app[14].

[12] After Giovanni Pico della Mirandola, an Italian 15^{th} century philosopher known for his prodigious memory. About twenty years ago, at my old workplace ORL, for a short while we used the abbreviation PiCO (note lowercase "i") to mean "Personal interactive Computing Objects" [40]; despite the striking similarities there is no intended connection between that name and this, and my Pico is an eponym rather than an abbreviation. Finally, Pico is also totally unrelated to the 10^{-12} SI prefix—and to the other thing they use that word for in Chile.

[13] With both usability and security implications.

[14] But see footnote 34.

But offering the correct password to the user isn't enough. If the password is strong (long, mixed-case and unpronounceable), even if I am no longer forced to remember that the password was `u4Hs9DB66Ab18GIUdCVi`, it is still rather tedious and error-prone to have to transcribe it[15]. With Pico, therefore, **No-Typing**: the user no longer has to type the damn password[16].

A rarely discussed problem of traditional password-based authentication is that, if a session lasts several hours[17], the app has no way of telling whether the prover is still present—it only knows that the prover was there at the start of the session. Repeatedly requesting the password after every few minutes of inactivity is hardly acceptable from a usability perspective, so the app must live with a window of vulnerability during which access is granted even though the principal interacting with the app may have changed. How much nicer it would be if the app could sense the presence of the authenticated principal continuously throughout the session, but without imposing an additional burden on the user![18] Pico uses short-range radio to offer the **Continuous** benefit: the user is authenticated to the app continuously throughout the session, not just at the beginning. In the course of the session, the app can lock and unlock itself automatically depending on the presence of the user's Pico (section 4.2).

Next to these usability benefits, Pico also offers several security benefits over passwords. Firstly, by letting the Pico, rather than the user, randomly[19] choose strong credentials for each account, we solve two typical problems of passwords: it is not possible for a careless user to choose a weak (brute-forceable or guessable) password (**No-Weak**); and it is not possible for a user to reuse the same credential with different apps (**No-Reuse**). Therefore it is not possible for a malicious verifier to impersonate the user elsewhere, or for a careless verifier (whose verification database gets compromised) to cause the user to be impersonated elsewhere.

Another problem is that a malicious app could masquerade as the genuine app and trick the user into revealing the password. With web sites this is commonly indicated as *phishing*, but an equivalent attack could also occur locally, for example with a Trojan application displaying a fake login screen on your computer. To prevent this, Pico authenticates the app before supplying the user's credentials, thus offering **No-Phishing**: it is not possible for a malicious app to steal the user's credentials by impersonating the app to which the user wanted to authenticate. The machinery for such authentication is available to web sites in the form of Secure Sockets Layer (SSL) but, given our pledge to Works-For-All, we want No-Phishing to apply in all other cases too.

[15] Thus by using passwords we impose at least three distinct usability burdens on the user: burden of **remembering**, burden of **selecting**, burden of **typing**. Pico avoids each of them with, respectively, Memoryless, No-Search and No-Typing.

[16] In fact the password isn't even there any more, but that's an aside.

[17] Sometimes even weeks, in the case of persistent login cookies.

[18] Section 6 discusses relevant prior art [40,20,21,6,25].

[19] Pico will need a good quality source of random numbers.

My minimum requirements for a password replacement system:	
Memoryless	Users should not have to memorize any secrets
Scalable	Scalable to thousands of apps
Secure	At least as secure as passwords
Additional requirements if token-based:	
Loss-Resistant	If token lost, user can regain access to services
Theft-Resistant	If token stolen, thief can't impersonate user
Benefits promised by Pico in addition to the above:	
(Usability-related)	
Works-For-All	Works for *all* credentials, not just web passwords
From-Anywhere	The user can authenticate from any client
No-Search	The user doesn't have to select the correct credentials
No-Typing	The user no longer has to *type* the damn password
Continuous	Authentication is continuous, not just at session start
(Security-related)	
No-Weak	The user cannot choose a weak password
No-Reuse	The user cannot reuse credentials with different apps
No-Phishing	Phishing (app impersonation) is impossible
No-Eavesdropping	Network eavesdropping is impossible
No-Keylogging	Keylogging is impossible
No-Surfing	Shoulder surfing is impossible
No-Linkage	Different credentials from same user can't be linked
Additional desirable properties that are not goals for Pico:	
No-Cost	As cheap to deploy as passwords
No-App-Changes	Deployable without changes to existing apps
No-Cli-Changes	Deployable without changes to existing clients
Also worth considering for a fair comparison:	
Implemented	This system was built, rather than just described
Open	The code and design are available as open-source
Widely-Used	Has been used by over a million individuals
No-Carry	Does not require the user to carry anything
No-TTP	No reliance on a TTP who knows your credentials

Table 1. Summary of desirable properties of password replacement systems. In the text, benefits are listed in boldface where first defined.

SSL also offers an additional protection: an encrypted channel that prevents network eavesdroppers from overhearing the password to reuse it later. Pico, too, offers the **No-Eavesdropping** benefit: it is impossible to extract any usable credentials by listening in (at any stage) between the Pico and the app.

Even with SSL, though, a keylogger may steal the password before it enters the encrypted pipe: maybe because the victim's machine has been compromised by malware or because it's not even theirs (cybercafé) and they don't know what's installed on it. With Pico, therefore, we offer **No-Keylogging**—a major benefit over passwords, meaning that your credentials cannot be stolen even if your local endpoint machine is compromised[20]. The low-tech equivalent of keylogging is shoulder-surfing, which passwords only partially avoid[21] by not echoing the typed characters, at the expense of usability. Pico, instead, as a consequence of No-Typing, also fully offers **No-Surfing**.

For privacy, we design the Pico so that an app (or even several colluding apps) cannot link the different credentials used by a Pico user: **No-Linkage**.

We also explicitly list some non-goals: Pico does *not* attempt to be as cheap to deploy as passwords (**No-Cost**), nor to be deployable without modifications—whether to verifiers (**No-App-Changes**) or to provers (**No-Cli-Changes**).

Finally, since we might wish to use the list of benefits in table 1 as column headings for a matrix comparing competing alternatives[22], it is only fair to list other benefits that Pico currently does not offer, much as it hopes to in the future—such as having actually been built (**Implemented**), having its design blueprint and source code published as open source (**Open**), having been used by over a million people (**Widely-Used**)—as well as other benefits that passwords themselves already offer but that some of their replacements don't, such as the usability benefit of not requiring users to carry anything (**No-Carry**, which Pico can't possibly offer) or the security benefit of not relying on a Trusted Third Party not controlled by the user (**No-TTP**, a benefit that Pico offers but which is violated by some in-cloud SSO password management schemes that deposit the passwords with an entity that then issues them to apps on your behalf).

[20] So long as the Pico itself isn't, that is. Conceptually, one might object, we have only moved the endpoint one level back. But it still makes a major difference in practice, particularly because the Pico is not a general-purpose computer running user-installable code, and therefore the task of securing it against malware is not as hopeless as it would be for the actual machine. Laurie and Singer [22] have a related discussion about their Neb.

[21] The finger movements can still be observed by a skilled operator. Not as scalable as keylogging, though, as it requires a human attacker.

[22] We fully agree with Herley and van Oorschot [16] that, to find suitable replacements for passwords, we need to specify the requirements clearly and then compare competing alternatives against these requirements. Cfr. footnote 74.

3 User authentication with the Pico

3.1 Core design of the Pico

The Pico is a small, portable, dedicated device, intended to be carried along all the time, just like your watch or home keys. It has two important buttons called "main" and "pairing"[23], a small display, a camera suitable for the acquisition of 2D visual codes and a short-range bidirectional radio interface. It could be shaped like a smart phone but also like a watch, a key fob, a bracelet or an item of jewellery.

Internally, the Pico's permanent memory (which is encrypted) contains thousands of slots, one for each pairing between the Pico and an app. Each slot contains the Pico's credential (a private key) for that particular pairing and whatever else is necessary for mutual authentication.

Each app, whether local or remote across the network, has its own public-private key pair. The Pico talks to the app over radio (either directly or through intermediaries) by sending an encrypted message to the app's public key (No-Eavesdropping). The Pico's message contains an ephemeral public key generated by the Pico, which the app can use to reply over radio[24]. The Pico remembers the public key of the app when it first pairs with it (section 3.3), thus defeating phishers and other men in the middle in subsequent interactions. This is conceptually similar to the Pico doing some kind of SSL with the app, but without a PKI to certify the app's key[25].

This arrangement gives us bidirectional encrypted wireless communication between the Pico and the app, but so far only the app endpoint has been authenticated. To authenticate the Pico to the app, the Pico proves ownership of the credential established during pairing (section 3.3) (No-Typing).

[23] Plus possibly other controls—maybe "soft" buttons based on a touch screen—to scroll the display etc. If we can get away without them, so much the better. The goal is for the UI to be minimal but without becoming frustratingly unusable; whether any given design achieves this goal can only be validated through user studies.

[24] The reason for using an ephemeral key rather than the Pico's long term credential for that app is to protect the user's privacy and prevent tracking: the user's identity is only revealed to the app *after* the app's identity has been verified by the user's Pico. When the Pico first talks to the app, by encrypting its message to the public key whose hash is in the chosen visual code, the app has not yet proved possession of the corresponding private key, so it could still be an attacker impersonating the app.

[25] But without a PKI, couldn't a malicious app impersonate the genuine one from the first time onwards? Yes, in the sense that the attacker could fool the Pico into believing that her fake app is the genuine one. No, in the sense that if the victim already has a relationship with the genuine app (e.g. has a deposit account at that bank) then the middleperson attacker won't be able to access those assets with the credentials fraudulently established between the fake app and the victim's Pico. This is because the visual code authenticator provided by the app out of band in that case (see section 3.3) is tied to the real app's public key (No-Phishing).

The general model of the app is that it requires a *(userid, credential)* pair, which it verifies against its stored validation data in order to authenticate the user. This suits most cases, including some in which the userid is implicit. There are, however, cases that this model does not represent well and that we must treat differently, as discussed at the end of section 3.4.

The opening screen where the app would normally display a "userid/password" request is now augmented with a 2D visual code (using the technique pioneered by McCune, Perrig and Reiter [24]) that encodes a hash of the app's self-signed certificate, containing among other things the app's human-readable name[26] and public key. Whenever that screen is visible, the user can initiate one of two actions by acquiring the 2D visual code by pointing the Pico's camera at it and pressing either of the Pico's buttons: the main button to send pre-established credentials to the app (as if typing a password), or the pairing button to establish a new pairing (as if creating a new account).

Transmitting the hash over the visual channel and the rest over radio constitutes a multi-channel security protocol [41]: most of the protocol takes place over a main bidirectional channel, namely radio, that is convenient to use and has good capacity and latency; but one message instead goes over an auxiliary channel that (despite being only unidirectional, requiring manual acquisition and being of limited capacity and latency) offers data origin authenticity and forces the user to actively designate the intended app.

3.2 Main button: Offer credentials

Pressing the main button of the Pico is the equivalent of typing the password. Which one? There is no password as such, but the Pico automatically selects the appropriate credential based on the visually acquired hash for the app (No-Search, but see footnote 34). If the app's public key is not among the ones stored in any of the Pico's slots, the process stops[27]. Assuming the Pico had not already been fooled during pairing, phishing is impossible (No-Phishing).

If the Pico recognizes the app, it talks to the app's public key over radio and sends it an ephemeral public key to reply to. The Pico challenges the app to prove ownership of the app's private key. Once the app does, the Pico sends its long-term public key for that pairing, thus identifying itself to the app, and then, as challenged by the app, proves ownership of the corresponding private key (for example by signing a specially-formatted message containing a challenge defined by the app), thus authenticating itself to the app.

[26] To have something to call it by on the Pico when necessary.

[27] Possibly with a phishing warning, as the user's intention suggests a belief that an account has already been set up with that app. Various heuristics could be used to assess the likelihood of the current app being a phish, including whether its human-readable name (or even page layout and colour palette) is similar to the one of any app registered in the Pico's slot (as in: did the user click the Main button without meaning to, or because she genuinely thought this was the banking site for which she already has a registration?).

Unlike a password, that can be keylogged when it is typed, here the credential is a secret key that never leaves the Pico[28]. The use of challenge-response, instead of merely exhibiting the raw credential, is how the Pico provides No-Keylogging.

Now the Pico and the app have mutually authenticated: they can set up a session key and proceed with continuous authentication (section 4.2).

Note that, despite the use of the visual channel, this authentication procedure is still vulnerable to a relay attack. Adding a nonce to the visual code, while useful, does not stop the attack because the "unique" visual code could be relayed as well[29] [30]. Stajano et al [39] discuss fancier and less practical multichannel protocols specifically aimed at thwarting relay attacks, but this isn't one of them. The quest for a reliable solution against relay attacks must continue[31].

3.3 Pairing button: Initial pairing

Pressing the pairing button is the equivalent of creating a new account.

We must distinguish two cases: pairing with an app that has no prior relationship with the user (as when creating a new account with a free webmail

[28] The Pico is not a general purpose computer, does not accept external code and could even include a small hardware security module for its key storage.

[29] Norbert Schmitz pointed out to me at Passwords^11, commenting on the case in which the Pico's radio talked to the local computer rather than directly to the app, that an attacker could visit the app's login page, show the victim the visual code of that app on a fake page, let the victim acquire it and unsuspectingly perform the radio part of the protocol with the real app on the attacker's computer nearby. At that point the attacker, sitting in front of the app's real page, would find himself logged in with the credentials of the victim. Quite right. As a countermeasure, Norbert suggested that the app embed a nonce in the visual code. This turns out to be necessary anyway if we let the Pico talk to the app without going through the local computer (e.g. via the cellphone network or wifi), if nothing else in order to distinguish concurrent login sessions from each other (see footnote 37); but it is insufficient to stop the relay attack, because the attacker could still relay the visual code with the genuine nonce too.

[30] Dirk Balfanz also pointed out the same attack at Usenix Security 2011 and showed me how Google addresses the same problem in their similar (experimental) system: the cellphone acting as the Pico shows an interstitial page asking the user to confirm that they really meant to log into Google. This solution works, but suffers from the well-known security-usability problem whereby a proportion of users will just become accustomed to clicking OK without really checking.

[31] In this context, distance bounding protocols based on timing of a speed-of-light message exchange (such as the Hancke-Kuhn [13] that we envisage using for the Picosiblings in section 4.1) cannot be used on the end-to-end link between Pico and app (which may go across the Internet and/or the cellphone network). Conversely, bounding the distance between the Pico and the terminal, rather than between the Pico and the app, would be hard, because the terminal is untrusted and shares no secrets with either the Pico or the app; but more importantly it would be pointless, as the terminal itself could be one half of a relaying attacker.

provider) or pairing with an app whose back-end already has some relationship with the user (as when signing up for online banking with a bank where you have already deposited some money).

In the first case, since we work without a PKI, a malicious app could in theory masquerade as the app that the user intends to pair with[32]. In the second case, at the time the "prior relationship" is established between app and user before the Pico first sees the app's "userid/password" screen, the app's back-end must give the user some special codes that will allow the user's Pico to recognize the app when it first sees that screen. One code will be the normal visual code of the app, encoding the hash of the certificate that contains the app's public key; another code will be a nonce, signed by the app, that identifies the user to the app, so that any resources (e.g. the money deposited by the user) can be associated with the user's Pico once that Pico presents that signed nonce during initial pairing. The Pico must thus have a procedure[33] for acquiring the signed nonce *in conjunction* with the app's visual code, with the semantics that the nonce must be sent only to that app, only during the initial pairing with that app. In either the first or the second case, once the initial pairing has taken place, the Pico will only ever send its credentials for that pairing to the app it originally paired with. This is how the Pico bootstraps its No-Phishing property.

Back to what happens when the Pico acquires the visual code of the app when the user presses the pairing button. If the Pico already knows that app (i.e. it has that app's public key in a slot), it may issue a warning: "you already have a pairing with this app, do you really want to establish another one?" But it would only be a warning, not a veto, because the user is still free to, say, open several webmail accounts under different names[34]. Assuming the user does wish

[32] But philosophically, even if we had a PKI, all the PKI could do would be to ensure that the public key of the app matched its human-readable name. It could still not guarantee that the user is really pairing with the intended app, as opposed to being fooled into pairing with an app with a similar-sounding name. From a theoretical viewpoint, the user needs *some* relationship with the app (even just knowing its correct name); otherwise she can't be fooled into pairing with the wrong app because there can be no right or wrong app by definition—no way to tell them apart. Reversing the argument, if we accept that the user needs initially to acquire an identifier for the app through some other trustworthy means, we might as well say she ought to use that unspecified trustworthy means to acquire the app's visual code (or the public key itself) with her Pico that first time—and thus we fall into the second case, which is safe.

[33] Possibly involving a third button. This is preferable over relying on tagging information in the visual code for the signed nonce, because otherwise an attacker could perhaps social-engineer the user into feeding a signed nonce to her Pico without realizing it, thinking she is just pairing with a new app.

[34] If the user maintains several distinct pairings with the same app, though, when pressing the Main button the Pico will not fully be able to honour No-Search and will have to ask the user, with some appropriate UI, to select the intended persona for that app. This is unavoidable, whatever the design, and not a failure of the Pico—it is not possible to select the userid and credential automatically and yet give the user the freedom to maintain several independent pairings with the same app. Note also

to proceed with pairing, the Pico gets the full public key of the app via radio, checks that the key matches the acquired visual code, possibly caches it in the Pico and, if all is well, responds with an ephemeral public key. As before, the Pico first requests proof that the app knows the private key matching the public key in the visual code before telling the app the Pico's own long-term public key for that account[35].

If the user had a "prior relationship" with the app then, now that a bidirectional encrypted channel has been established, the Pico sends the app the signed nonce, so that the app can associate the newly received credential with the already-existing back-end account. If appropriate, a userid is also defined on the app using the traditional procedure (on the host computer, not on the Pico) and sent back from the app to the Pico to be stored in the relevant slot[36].

3.4 Replacing all passwords

Web apps The case that has received the most attention, both in the literature and in actual implementations, is that of authenticating a user to a web app. Partly because it's web apps that are primarily responsible for today's proliferation of passwords and partly because the uniform interface offered by the web makes this case easier to address, for example by augmenting the browser or by using a proxy. However, as we said, our aim with Pico is to get rid of all passwords, not just web ones, so we should be more general in our design. The app requires the ability to talk to the Pico over the two channels (radio and visual) of our multi-channel protocol, but these channels don't necessarily have to go through the browser. While it is easy for the visual channel to go through the browser (it's just a matter of rendering a bitmap as part of the web page), the other channel could be implemented as radio only from the Pico to a piece of middleware on the local computer, and from there turn into IP packets and reach the relevant app across the Internet without even touching the browser[37]. It would even be conceivable for the Pico to talk to the app via an external radio access point rather than via the computer on which the app's visual code is displayed.

that the Pico's design, without PKI and with independent user credentials for each pairing (as opposed to, say, using a global key pair for the Pico), protects the user's privacy with respect to the app. Even several userids of the same Pico with the same app will appear to the app as totally independent, offering No-Linkage.

[35] Which may have been precomputed, to save time during pairing.

[36] This userid is presented by the Pico to the user when, for example, on pressing the Main button, the Pico asks the user which of several accounts with this app should be used to log in.

[37] If the web app added a nonce to each rendering of the visual code and asked the Pico to sign and return it in the rest of the protocol, it would know which "page impression" triggered which instance of the radio subprotocol, even if it didn't arrive from within the browser.

Other remote apps—perhaps non-graphical Having factored out the browser, there are no major architectural differences between the case of web apps and that of other remote apps (mail and various forms of network login). The only notable point is that some apps (e.g. ssh) are text-based and it might be difficult or impossible for them to display a visual code[38] at the time of requesting the "password". In such cases, so long as an alternative pairing procedure can be set up (for example using a web client) in which the app's back-end can securely communicate the app's public key to the Pico, then subsequent authentications will still be protected from phishing. The user interaction flow will have to be designed carefully, though, because there will be no visual code to acquire when pressing the Main button. The app's public key will be available over the radio channel, but so would several others, potentially. In absence of the visual channel, the Pico will have to ask its user to confirm which of the recognized public keys (if any) it wishes to send credentials to, thus partially reneging on No-Search.

Local apps—and protection of app secrets As far as Pico-based authentication is concerned, there are also no major differences between remote apps and those running locally, such as logging into a local account on your computer, running an application that requires administrator privileges, switching to another user, unlocking the screen saver or entering the restricted section of the BIOS configuration utility[39]. The main difference is that the app's secret must be stored on the local computer.

On the back-end side, whether local or remote, apps using passwords can be attacked, depending on the strategy they use to store the verification credentials. If passwords are stored in cleartext[40] then a breach into the back-end exposes them wholesale, regardless of their strength. If they are stored in hashed form[41], a breach into the back-end exposes them to brute-force and dictionary attacks, which can be mitigated by increasing the number of rounds of hashing (to counter Moore's Law) and by salting (to thwart precomputation). But stolen passwords are only valuable if they were common to several accounts and thus can be reused elsewhere, or if the site owners don't notice the breach and revoke them[42]. With Pico, credentials are never reused across accounts (No-Reuse) and stealing the verification data (the Pico's public key for that account) only allows the

[38] In theory we might try to provide an alternative implementation of the visual channel by creating an ASCII marker (with suitable redundancy) and performing OCR on it. In practice such a solution is unlikely to match the robustness and capacity of regular 2D visual codes.

[39] Note that the last two typically don't require a userid.

[40] Irresponsibly careless but not unheard of—cfr. RockYou, December 2009.

[41] But perhaps unsalted, as still happens in Microsoft's NT LAN Manager (NTLM, still widely deployed in 2011, even on new systems, for compatibility reasons), thus opening the door to precomputed brute-force attacks such as Rainbow tables.

[42] And, in the latter case, the attackers have already broken into the back-end to steal the passwords, so how plausible is it to assume that they cannot also directly steal the assets that these credentials are meant to protect?

attacker to talk to the Pico and verify signatures made by the Pico, not to impersonate the Pico—not even to the app from which the verification data was stolen[43]. What *could* be stolen is the secret keys of the app (rather than of the various Pico clients who authenticated to the app), which could then be used to mount phishing attacks against any users of the app, indirectly defeating the No-Phishing claim. One possible defense against that threat, if warranted by the value of what the credentials protect, is to store the app's secret key in a Hardware Security Module (HSM) on the verifying host[44]. This countermeasure can be applied to both local and remote hosts.

Non-computer apps Authenticating to a non-computer app covers such cases as having to type an unlocking PIN into a car stereo, a burglar alarm, a cash dispenser, an electronic safe and so forth.

Under the assumption that some form of bidirectional channel is available between the app and the Pico[45], this case is not substantially different from the one already seen for non-graphical apps. An alternative pairing procedure is needed whereby the public key of the app can be securely transferred to the Pico. Of course, if the device has a graphical display, as more and more will, it can instead show its visual code just like a computer-based app.

The case of the cash dispenser is interesting: if the PIN is used to authenticate to the bank, as in the case of magstripe cards, the app is either the ATM (if the PIN check is local) or the issuing bank (if the PIN check is done at that bank). But if the PIN authenticates the user to a chip in the smart card, it's that chip itself that is the app, and the ATM is only its display[46] and radio interface. In any case, since the Pico system architecture is end-to-end between the Pico and app, we don't really care that much about how many intermediaries we have.

Passwords not used for authentication Consider the case of a program that performs symmetric encryption and decryption on files. For sure it requests you to remember and type a password (potentially a different one for every file) and as such it must be catered for by the Pico, otherwise we'd be failing to offer the Works-For-All benefit.

But it is difficult to model it as a login-style app requesting userid/password, as we have done so far. Would every file be a different app? Or would the encryption program be the app, and every file a different userid? What if I wanted to encrypt the same file under five different keys, to give it to five different

[43] If the attackers who can steal the credentials can also rewrite them, they might be able to substitute their own public key to that of the victim in order to access the victim's account; but the comment in footnote 42 applies.

[44] And, once its secret key is safe, the app can sign the public keys of all the Pico clients it pairs with before storing them, to preserve their integrity.

[45] Via local short-range radio or perhaps a radio-to-IP access point if the app already has IP connectivity.

[46] But note how the visual code for the card's chip could even be printed on the card.

recipients—should the userid be my own annotation, such as "file X that I'm sending to recipient Y"?

And a more fundamental problem: even if my Pico successfully proved ownership of the correct credential for the chosen file, how would the app decrypt the file for me? Would it authorize use of a decryption key stored next to the verification data for my credential? This additional indirection is architecturally wrong because then the app holds the key to my file. My file is no longer protected by encryption but by the (probably much weaker) mechanism through which the app protects that key. If the app is compromised, my file is no longer secure. That's not how things should work.

There is indeed a crucial conceptual difference between this case and the previous ones: the file encryption application, even though it asks for a password like all the others, is not *authenticating* the user, nor is it *verifying* any credentials whatsoever: it is just *accepting* a passphrase from the user and deriving from it a cryptographic key with which to decrypt the selected file. This app, unlike the others, shouldn't even *know* how to unlock the user's assets (i.e. the encrypted file) without the user-supplied string—it's not the case that the app has access to the data but won't grant it to others until they prove they're worth it. And it's crucial that the app *not* be able to decrypt—the information necessary from decryption must be supplied by the user, not remembered by the app. It's a subtle but significant distinction[47].

For this kind of application, the proper thing for the Pico to do is to use the previously-described machinery just to establish a secure channel with the app (to ensure No-Eavesdropping and especially No-Keylogging), but to then transmit a strong encryption key to the app over that channel. Some additional user interface elements on the Pico beyond the Main and Pairing buttons will have to be involved, and some careful thought will have to go into the design of the user interaction flow. As a first sketch: the user starts by authenticating to the encryption app using the Pico; then the UI of the app is used to select a file and the operation (encryption or decryption) to be performed on it. For encryption, the Pico creates a random bit string of the appropriate length and sends it to the app as the key to be used. For decryption, the user selects the file using the app's UI and the Pico provides the correct decryption key for it[48].

The case of passwords not used for authentication is somewhat annoying because it requires special treatment, but it is also the most intellectually stim-

[47] Especially if you think about the one-time pad where different decryption keys yield different plaintexts for the same ciphertext, all equally valid a priori

[48] This problem is not trivial, and as yet unsolved, if we are also to provide the No-Search benefit of not requiring the user to pick the appropriate credential on the Pico itself. Using the file name to identify the password is ambiguous—the user might have several files by that name, in different places or at different times, to be encrypted with different keys. Using the hash of the encrypted file is only appropriate if the file is immutable once encrypted; otherwise, if the file is updated and re-encrypted, its key would change, causing trouble if we ever had to access previous versions (e.g. when restoring from a backup).

ulating. The open question is whether there are any further uses of passwords that are not yet captured by any of the interaction models described above.

4 Details of Pico operation

4.1 Locking and unlocking the Pico with the Picosiblings

In any token-based authentication system, all the verifier can do is to check for the presence of the token; from that, it is a leap of faith to infer the presence and consent of the actual user.

Many token-based systems quickly dismiss this problem, either by equating it to that of your physical home keys (i.e. by doing nothing and just hoping you won't lose the token) or by simply protecting the token with a short PIN. A PIN with an HSM-enforced rate limit after three wrong guesses might appear to offer reasonable security; but that's before taking several factors into account: people's tendency for choosing easily guessed PINs (violating No-Weak) and/or the same PIN as for other apps (violating No-Reuse), the cost of providing a PIN pad on the token[49], the burden and security risk of having to type the PIN into the token (violating No-Typing and No-Surfing) and of course the burden of having to remember a PIN in the first place (violating Memoryless).

Then there is the window of vulnerability from time of check to time of use, now shifted one level from authentication of the token to the app to authentication of the user to the token: user types PIN in the morning, unlocking token for the day, then loses token at lunchtime, allowing finder to use it in the afternoon. Requesting the PIN more frequently has a high usability cost.

For all these reasons, the Pico uses its own method (the Picosiblings, described next) for locking and unlocking. The rest of this section (4.1) can be considered conceptually separate from the core design of the Pico described in section 3.1 but the discussion above shows how many of the benefits promised by the Pico (Memoryless, No-Typing, Continuous, No-Weak, No-Reuse, No-Phishing, No-Eavesdropping, No-Keylogging, No-Surfing) also crucially depend on the way in which the token itself is locked.

As already mentioned, the secrets of the Pico are themselves encrypted[50]. They are unlocked by the Pico Master Key, which is reconstructed using k-out-of-n secret sharing [35]. The shares are held by other entities known as the **Picosiblings**. The idea, which we first mentioned in 2000 as "family feelings for the Resurrecting Duckling" [36] and that was also independently suggested in 2001 by Desmedt et al [7], is that the Pico will "feel safe" and unlock itself when in the company of its Picosiblings; and defensively lock itself up otherwise.

The shares, except for the two special ones described next, are held by the Picosiblings, which are small objects chosen for the property that the user will wear

[49] Or, if the PIN pad is provided externally, as in EMV payment cards, the possibility that an attacker might intercept the traffic from the keypad to the token, or insert malware on the keypad itself (violating combinations of No-Eavesdropping, No-Keylogging, No-Phishing) [9,2,5].

[50] With the equivalent of "full disk encryption" or perhaps even with an internal HSM.

them practically all the time: glasses, belt, wallet, various items of jewellery—even piercings, wigs, dentures and subcutaneous implants. In daily usage, the user interacts with them in no other way than by wearing them: they talk to the Pico with short-range radio and they don't require much of a user interface.

With an appropriate initialization protocol based on the Resurrecting Duckling [38], each Picosibling is securely paired to the Pico. From then on it responds to the radio enquiries of its master Pico using a "Picosibling ping" protocol with the following properties:

- The Pico can ascertain the presence of any of its Picosiblings in the vicinity.
- The Picosibling responds to its master Pico but not to any other Pico.
- At each ping, the Picosibling sends its k-out-of-n share to the Pico, in a way that does not reveal it to eavesdroppers.
- An eavesdropper can detect the bidirectional communications between Pico and Picosiblings but not infer identities or long-term pseudonyms.
- The Pico can detect and ignore old replayed messages.
- The Pico can detect and ignore relay attacks (e.g. with Hancke-Kuhn [13]).

Internally, the Pico keeps an array with a slot for each paired Picosibling. Each slot contains, among other things, a decay (countdown) timer and some space for the key share contributed by the Picosibling. At each valid ping response from a Picosibling, the corresponding decay timer is refilled with the starting value (e.g. 1 min) and starts counting down immediately; the Picosibling's share is refreshed and, if appropriate, the k-out-of-n secret is reconstructed. Whenever a decay timer expires, the corresponding share is wiped; the k-out-of-n secret is also wiped and, if possible, reconstructed from the other shares.

When the shared secret is not available, the Pico's credentials are inaccessible and so the Pico doesn't work. If the Pico fails to reacquire k shares within a set timeout (e.g. 5 min), it switches itself off, thus forgetting *all* shares, and must be explicitly turned on and unlocked before it will work again[51].

Two of the shares are special[52] and have a much longer timeout (e.g. a day). One of the special shares is derived from a biometric measurement[53], with suitable error correction [14]. The purpose of this share is to ensure that, even if an attacker gains control of enough Picosiblings (for example by raiding the user's

[51] The user may of course also switch off the Pico intentionally even while all the Picosiblings are in range.

[52] Depending on the chosen policy, they might even weigh more than "1" in the k-out-of-n budget. Their presence might be required regardless of that of any others.

[53] The biometric as an additional authentication factor has the advantage of usability (and few of the privacy concerns normally associated with biometric authentication, because the verifier is your own Pico rather than Big Brother) but won't be as strong as the published statistics on biometric authentication reliability might suggest because here the verification is not supervised by a human verifier suspicious of the prover. The verification process thus isn't resistant to an adversary who has control of the Pico and feeds it iris photographs or gummy fingers [23]. The process should still make at least a basic attempt at verifying that the biometric is live.

swimming pool locker), a lost Pico will eventually switch itself off if it is away from its owner.

The other special share is obtained from a remote server (conceptually belonging to the user) through a network connection. It has a dual purpose: auditing the reactivations (the server keeps a log of where and to which address it sent out its share) and allowing remote disabling of the Pico (the user who loses control of the Pico can tell the server not to send out the share any more[54]).

How does the user manage her Picosiblings? Aside from the two special shares, handled separately, let's say that the security policy of our user requires proximity of three other Picosiblings to unlock the Pico, for example two earrings and a pair of glasses. Even though only three items are necessary, the user ought to register several pairs of earrings and several pairs of glasses[55] and perhaps also other items for good measure, such as a bracelet, a belt, a medal, a smart card and so on[56]. The n in k-out-of-n secret sharing says how many Picosiblings are registered with this Pico; it can be much greater than k, which says how many Picosiblings the Pico must sense to unlock. All the unworn $n - k$ Picosiblings must be kept in a safe place. Defining a new set of n Picosiblings, as well as changing n and k, is conceptually an atomic operation (see section 4.3).

4.2 Continuous authentication

A major advantage of Pico is its CONTINUOUS benefit: once you have "logged in" by acquiring the app's visual code and letting the Pico do its stuff over the radio, the app can continue to ping the Pico over their confidentiality- and integrity-protected channel and use that to confirm that you and your Pico are still around. As soon as you are not, the app can block access[57]; and, what's even better, if your Pico reappears before a set timeout expires, the app can grant access again without even asking you to reacquire the visual code.

One potential threat in this situation is the relay attack, which an adversary could use to make it look as if you were still close to the computer even after you left. Guarding against relay attacks at the level between Pico and app is harder than at the level between Pico and Picosiblings because the link may have unpredictable latency, possibly even going through the Internet, and thus distance-bounding methods won't work reliably.

Further engineering details to be addressed include state preservation (things must work like an automatic suspend/resume where you still find everything just

[54] Ironically, the user will need a way to issue this order without her Pico. One solution, inspired by the work of Schechter et al. [34], might be to control the remote server with the secret-sharing-based consent of a few trusted friends.

[55] So that she can still use her Pico whatever earrings she chooses to wear today.

[56] So that she can use other items on days when she wants to wear no earrings, or non-Picosibling earrings. And also so that she has enough extras that, even after losing today's earrings and glasses or having them stolen, she can still unlock her Pico, or her backups (see section 4.3) if she also lost her Pico.

[57] Like a locked screen saver that came on *only* when you left, not merely when you stopped typing for a while; and *as soon* as you left, not merely half an hour later.

as you left it, not like an automatic logout/login where all open programs get closed) and the nesting of app sessions (one of the apps may be your actual login session on the local computer and another might be your webmail session within your web browser; each with its own key pairs and so on; the suspend and resume pairs must nest in the correct order).

4.3 Backup

To protect the user from permanent loss of access if the Pico is destroyed or lost (Loss-Resistant), regular backups must be taken—but actual users never do backups. We therefore introduce what Norman [26] calls a forcing function: the Pico's docking station recharges your Pico (as you do with your phone) and automatically takes a backup every time you dock. Since the storage of the Pico is encrypted, the backup is too, automatically. The docking station works as a standalone device, writing the backup to a memory card (and possibly also storing the last few internally)[58] but it can also be connected to a computer to provide a user interface. An encrypted backup can be restored onto a virgin Pico, but it only becomes accessible by unlocking the Pico as usual through its Picosiblings (section 4.1).

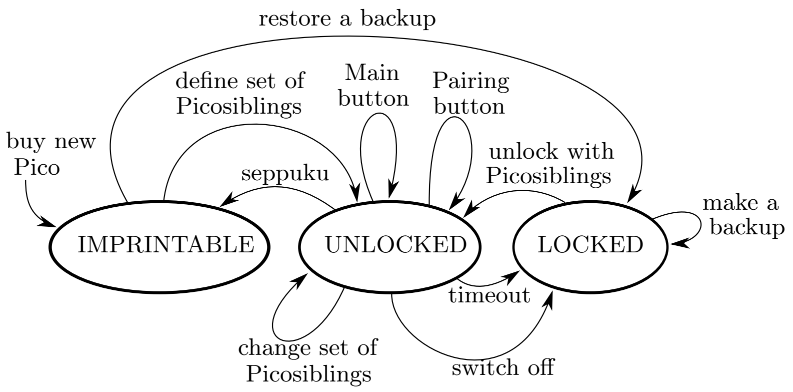

Fig. 1. State diagram of Pico, based on Resurrecting Duckling.

More formally, the model for the Pico is derived from the Resurrecting Duckling [37,38]. With reference to the state diagram in figure 1, a brand new Pico

[58] The docking station contains no secrets and is not paired with the Pico or Picosiblings. Even so, since it contains the backups, it is prudent to leave one's docking station at home ("safe place") and use a different one for trips, not for confidentiality but for availability, i.e. so as not to lose the only copy of the backups. It would in theory be possible to use the same docking station everywhere if one religiously extracted the backups and stored them elsewhere, but Murphy says it's unwise to rely on this happening.

is in the "imprintable" state and contains no credentials. The other two states, "unlocked" and "locked", both count as imprinted. From "imprintable" you can either restore an existing backup, thus taking the Pico to the "locked" state, or define a new set of Picosiblings and pair them with the Pico[59], taking the Pico to the "unlocked" state.

The "unlocked" state is the one in which the Main and Pairing buttons work. From there, you move into the "locked" state by timeout or by explicitly switching off the Pico. From the "unlocked" state you can change the set of Picosiblings[60]. You may also order seppuku (suicide), thus wiping all stored credentials and returning both the Pico and all its paired Picosiblings to the "imprintable" state.

The "locked" state is the one in which the Pico remembers credentials but won't let anyone use them. From this state you can take a backup (remaining in this state) and you can get back to "unlocked" with the cooperation of enough Picosiblings.

This model allows the possibility of having different Pico instances containing the same credentials—if they were all restored from a common backup. They might then go out of sync if credentials are added to one and not the other. Additional protocols might be defined to keep such instances in sync or to kill off all but one of them, relying on a hardware serial number and on the fact that all these Pico instances talk back to the network server that holds one of the special Picosibling shares (section 4.1).

It is also in theory possible to extend this model by allowing users to select and export individual credentials from one Pico to another. So long as this is done securely and the interaction is designed without excessively complicating the management interface, this feature might be useful to implement *selective delegation* ("I give you a copy of the key to my filing cabinet, but not a copy of my front door key"). It is architecturally cleaner to support this function at the verifier's end instead ("henceforth, Bob's key too will open Alice's filing cabinet"), but transferring the credential provides *plausible deniability* (now nobody can tell whether the filing cabinet was opened by Alice or by Bob)—which, depending on circumstances, might be a bug or a feature.

To recover after losing the Pico, the user obtains a new imprintable Pico[61], restores the backup onto it (bringing it to the "locked" state), then unlocks it with the Picosiblings. In the unlucky case that the user also lost the Picosiblings, she must unlock the backup using the spare Picosibling she wisely stored in her

[59] This is a complex subprotocol in which a random master key for the Pico is generated and then split into shares, and the shares distributed to the imprintable Picosiblings.

[60] This is another complex subprotocol in which existing Picosiblings are returned to the imprintable state, a new Pico master key is generated and the Pico is reencrypted under it, then the new master key is split among a new set of imprintable Picosiblings and the shares distributed to them; all somehow atomically, so that you can't lock yourself out of your Pico if something goes wrong halfway through.

[61] We can buy pay-as-you-go cellphones for less than \$10 nowadays, so imagine that to be the price of the Pico once everybody has one. People might even keep a blank imprintable Pico at home, and another at work, "just in case".

safe deposit box[62] when she defined the original set of Picosiblings. Once her restored Pico is unlocked, it would be a good idea for her to define a new set of Picosiblings, including some new ones to replace[63] the ones she lost, and then take a new backup.

4.4 Escrow

The Pico model is quite flexible and covers a wide spectrum of requirements, from those of the individual to those of the corporate user. It is possible, though by no means mandatory, to register additional Picosiblings and leave them in escrow with the company to allow access to the resources protected by the Pico if the employee is run over by a bus. Similarly, it is possible for an individual to register additional Picosiblings and leave them in escrow with a spouse, a trusted friend or a notary as a kind of "digital will".

It is also possible for a single user to operate more than one Pico[64], for example one for corporate secrets and one for private ones, so as not to have to place one's private secrets in escrow with the company.

4.5 Coercion resistance

The paranoid threat model for passwords is that you may be kidnapped by the mafia or the secret police and tortured in a dark basement until you reveal them. If you succeed in destroying your Pico before capture, then all your credentials become inaccessible and Pico removes the rational incentive[65] for torturing you.

If you are James Bond on a special mission and suspect that you might be captured[66], you might wish to trade off some availability in exchange for security and disable the remote server, after setting the timeout of the remote share to the duration of your trip. You won't be able to reactivate your Pico until you get back home if it ever goes off, but neither will your captors. Now all you need to do while being captured is to switch off the Pico, not even to destroy it.

The assumption is that your backups (and spare Picosiblings, and network server) are in the proverbial "safe place" and that if the bad guys allow you to go back to your safe place to get them, then they can't force you to come out of it. The bad guys could however force you to ask someone else to get the backups out of the safe place in your stead while you are still in their hands.

[62] Or gave to non-colluding friends, since Picosiblings naturally support secret sharing.
[63] And thereby implicitly revoke.
[64] Though they might each require their own cloud of Picosiblings. Since each Picosibling can only be paired to one Pico at a time, this puts a practical limit on the number of Pico per person, which is just as well to avoid the "I won't have the right Pico when I need it" problem.
[65] Of course they might still torture you out of revenge, or just because they like doing it—no guarantee they'll be rational about it.
[66] Or, more mundanely, if you are on a trip away from home and think there's a higher than usual risk that you might lose your Pico.

Nothing in this subsection is to be taken as much more than poor cryptogeek humour[67], but the semi-serious point is that it is possible to make the Pico a coercion-resistant system *if* you are prepared not to be able to recover its secrets yourself. If there is a way for you to recover from the destruction of your Pico, they might be able to force you to use it. But if you destroy the Pico and have no backups, they can torture you all they want but won't be able to recover your secrets[68], even with your full cooperation. You can't claim that for passwords.

Another practical strategy for protecting especially precious credentials (and for being able to deny that they even exist) is to dedicate a separate Pico just to them. Your ordinary Pico would hold your day-to-day credentials; then another special Pico (that nobody would know about, kept in a safe place and less likely to be misplaced or stolen) would hold the more serious and rarely accessed stuff. The caveat in footnote 64 against Pico proliferation applies, but having a separate "travel" Pico still sounds like a sensible precaution.

4.6 Revocation

To complete the Pico design we need appropriate facilities for revoking key pairs, both on the Pico side and on the app side since the authentication is mutual. We may well adopt the standard technique (used, among others, by PGP) of keeping in a safe place a revocation request signed with the private key to be revoked; but we'll have to address the standard problem of the intended party not hearing about this revocation in a timely fashion, especially in our decentralized non-PKI-based environment.

5 Optimizations (as Roger Needham would call them)

"Optimization", Roger Needham famously remarked, "is the process of taking something that works and replacing it with something that almost works, but costs less". There are various ways in which we could "optimize" the design of the Pico in an attempt to bootstrap its acceptance and market penetration[69].

[67] The *Akevitt* Attack, suggested by Erlend Dyrnes, is more effective and less violent: generously offer victim twenty shots of said Norwegian liquor; then, once victim under table, grab and use their still-unlocked Pico and Picosiblings. Perhaps one of the Picosiblings should include a mercury switch and stop working when the user is horizontal—but this would not suit Cory Doctorow who wants to websurf in bed.

[68] The main difficulty will be convincing them that that's the case before they start. "Read this before torturing me" might not work—especially with a paper this long.

[69] Once a critical mass of users is reached, these "optimizations" should be rolled back. The next concern will then be our ability to use Pico in absolutely all situations, including by users with disabilities (at this stage Pico seems to be doing fine, but this would have to be assessed more carefully) and in locations that disallow the use of a camera, such as certain military or industrial facilities (it's debatable whether we should change the Pico or choose not to support these cases).

5.1 Using a smart phone as the Pico

One of the "costs" of the Pico for the user, besides the price tag, is the burden of having to carry yet another gadget. To counter that and offer Quasi-No-Carry, some will want to implement the Pico as software for a smartphone, which already has a good display, a visual-code-compatible camera, a radio interface and, above all, is a device that users already carry spontaneously. But the smartphone is a general-purpose networked device and thus a great ecosystem for viruses, worms, trojans and other malware. I, for one, would not feel comfortable holding all my password replacements there. I also doubt that a smartphone program would be granted enough low level control of the machine by the OS to be able to encrypt all of its data without inadvertently leaving parts of it in places that an attacker with physical control of the device could re-read.

A halfway-house between an insecure smartphone and a secure piece of dedicated hardware and software might be to use your old smartphone as your Pico[70] and your new one as your regular phone. Not as secure as a purpose-made Pico, and it would fail to offer No-Carry, but it would be a cheaper way to prototype and field-test the Pico without building custom hardware.

5.2 Typing passwords

The most significant "cost" of the Pico is undoubtedly the fact that it requires changes to the apps; next to that is the cost of the changes (hardware and software) on the local computer. A way of eliminating these costs and thus achieving No-App-Changes and No-Cli-Changes is to drop public keys and visual codes and to make the Pico just remember straightforward passwords. With a USB connector the Pico could emulate a keyboard and type the password for you, honouring the No-Typing benefit[71].

The ways in which this optimization works only "almost" are unfortunately many. Giving up the SSL-like operation opens the door to app impersonation, losing No-Phishing. Because the app no longer has a visual code or public key, the Pico can't select the appropriate credentials, losing No-Search. Because the app can no longer ping the Pico, we lose Continuous. Passwords typed by the Pico can be intercepted, losing No-Keylogging. All we retain, provided that we manage the passwords sensibly, are Memoryless, No-Weak, No-Reuse, No-Typing and Quasi-Works-For-All[72].

An even more radical cost saving measure would drop the USB connector, forcing the user to transcribe the passwords manually and essentially bringing us back to the PDA with password wallet software. Neither option is recommended.

[70] Wiping it of any other apps and restricting network functionality to getting the special share from the remote server as described in section 4.1.

[71] Some commercial devices already do that—see footnote 83.

[72] If we claim No-Cli-Changes we must exclude the situations, like ATMs and burglar alarms, that don't already support external USB keyboards.

5.3 Removing fancy features

Further savings could be obtained by dispensing with some of the Pico's special features: by not bothering with the Picosiblings (section 4.1), perhaps replacing them with a master PIN; by dropping proper mass storage encryption (section 4.1); or by not implementing the automatic backup from the docking station (section 4.3). We noted at the start of section 4.1 the disadvantages of securing the Pico with a master PIN; and not performing backups automatically would be a *really* bad idea. On the other hand, future user studies might tell us that managing the Picosiblings is found to be as complicated as having to remember and juggle dozens of strong passwords. We need prototypes and impartial user studies to understand whether this is so. Picosiblings are just *one possible way* of locking and unlocking the Pico and as such they are the feature subset that one could most easily remove. Perhaps the first simplification might be to remove the special Picosibling share of the network server, which carries its own management burden; and then to remove the other special share (biometric). A more drastic alternative might be to omit the Picosiblings altogether.

5.4 Gradual adoption

Some users might welcome the convenience of the Pico but not trust it for absolutely everything: they might feel safer remembering one or two strong passwords for their highest security accounts and using the Pico for everything else. This is fine: until the Theft-Resistant and Loss-Resistant claims of the Pico have been validated beyond reasonable doubt by hostile review, we would consider this trade-off[73] a prudent diversification strategy.

6 Related work

Replacing passwords is an extremely crowded research area and any survey, however long, will necessarily be incomplete; for more references than we discuss here, start with the extensive bibliography in Bonneau and Preibusch [4]. We mentioned in the introduction Adams and Sasse's classic study on password usability problems [1], but in this section we focus instead on alternatives and solutions, highlighting how they score against the criteria set out in Table 1[74].

How do ordinary people cope with password overload? Following a long-established tradition many of them, having exhausted what Beautement et al. [3] would call their compliance budget, achieve benefits Memoryless and Works-For-All simply by writing their passwords down; some even keep them in

[73] The price to pay for not putting all the eggs in one basket is not just the downgrade from Memoryless to Quasi-Memoryless but also the loss of all the Pico's usability *and* security benefits over passwords for those high security accounts, including protection against keylogging and phishing.

[74] For reasons of space we only present significant highlights, rather than a complete (products $\times$ benefits) matrix, as some cells would require extensive discussion. But we might one day extend this section into a full stand-alone comparative survey.

a file on their computer, to allow cut and paste and thus enjoy NO-TYPING. A negligible minority of crypto-geeks do the same but also encrypt the file, thus degrading to QUASI-MEMORYLESS but additionally achieving SECURE and THEFT-RESISTANT[75]. This rather sensible approach was at some point embodied into stand-alone "password wallet" applications, of which one of the earliest was Schneier's Password Safe (1999). Password wallets running on PDAs rather than on the user's main computer were less at risk from malware, especially if the PDA wasn't network-capable, but gave up on NO-TYPING. Some of these programs also offered NO-WEAK and NO-REUSE, to the extent that the program could, on request, generate a random password. A significant advance in this direction became the integration of the encrypted password wallet within the web browser—one of the first examples being the Netscape/Mozilla family (1999?). This solution added NO-SEARCH to NO-TYPING, with a marked improvement in usability, but obviously gave up on WORKS-FOR-ALL because it only worked with web apps. The latest versions of such browser-based password wallets offer the option of storing the passwords in the cloud[76] and of synchronizing them among different browsers, to offer FROM-ANYWHERE; but some of these implementations are less convincing than others with respect to the trust that the user must place in the wallet provider. There exist also versions of password wallets integrated in the operating system rather than in the browser, such as OS X's Keychain. While they still fail to provide WORKS-FOR-ALL, they handle additional passwords not managed by the in-browser wallet, such as the ones for WiFi. Note how most of these systems only provide QUASI-MEMORYLESS rather than MEMORYLESS, because they require a (keyloggable) master password.

A conceptually related approach (having a single password that unlocks your other passwords that are then supplied automatically to the relevant apps) is the WIDELY-USED Single Sign-On (SSO). The useful taxonomy of Pashalidis and Mitchell [29] classifies SSO systems along two dimensions: local vs proxy (does the entity to which the user authenticates, known as the Authentication Service Provider or ASP, reside on the local computer or in the network?) and true vs pseudo (is SSO supported by design or just by transparent simulation of pre-existing authentication methods?). According to this taxonomy we might describe password wallets as local[77] pseudo-SSO systems. SSO systems may offer NO-PHISHING, NO-EAVESDROPPING and even, depending on the mechanism chosen for authenticating to the ASP, NO-KEYLOGGING (cfr. the IMPLEMENTED, OPEN "Impostor" [30], which uses a primitive challenge-response;

[75] Plus LOSS-RESISTANT if they back up the file.

[76] An interesting twist is a wallet that *generates* your passwords rather than storing them: `www.passwordsitter.de` derives your passwords by encrypting the app's name and other details with your master password, therefore weak and reused passwords are replaced by app-dependent hashes. It can work as an in-cloud wallet or as an offline one. However, while it's true that "your passwords are not stored", you aren't much better off than if they had been stored encrypted because, for your convenience, their server stores a file with your app names and the other relevant details—so your only protection is still the (keyloggable and potentially guessable) master password.

[77] Or proxy for in-cloud ones.

its Implemented successor "KYPS" [28] and the independently Implemented "URRSA" [10], which use one-time passwords). Pseudo SSO systems even offer No-App-Changes: the apps don't even know that the user is logging in through an ASP rather than directly. True SSO systems effectively get rid of passwords with respect to the downstream apps, thus offering No-Weak and No-Reuse and, with a proper protocol, No-Phishing, No-Eavesdropping, No-Keylogging and No-Surfing over the connection from ASP to app; but they might offer only "quasi" versions of these benefits if the user still authenticates to the ASP with a password. One further dimension that could be added to the Pashalidis-Mitchell taxonomy would distinguish between an ASP under the control of the user ("privacy-protecting"), which would also offer No-TTP, and one run by a third party ("privacy-invading", because the ASP would get to know about every app you visit)[78]. Both Pico (seen, at a stretch, as a kind of "local, true" SSO) and Impostor [30] (a "proxy, pseudo" SSO) would count as "privacy-protecting", whereas systems such as Facebook Connect or OpenID would be "privacy-invading". In this sense KYPS is "privacy-invading" but, in a strange combination, offers No-TTP because even physically capturing the server does not yield the stored credentials, which are reconstructed on access by XORing them with the OTPs.

One advantage of the Pico over SSO systems is Continuous, of which the Active Badge Implemented by Want and Hopper [40] offered an early glimpse as far back as 1992: it would lock the workstation's screen as soon as the user left. A more security-conscious approach to the problem came in 1997 when Landwehr [20] proposed, patented (with Latham [21]) and Implemented a system that would continuously monitor the presence of an RFID token worn by the user and, in its absence, disconnect the keyboard and monitor from the computer. To address the fact that a physical attacker could still access the raw disk, in 2002 Corner and Noble [6] presented (and later refined with Nicholson [25]) "Zero-Interaction Authentication", a well-engineered Implemented system in which proximity of the token would unlock the keys of the laptop's cryptographic file system[79]. Absence of the token would flush the decrypted files from the cache and wipe their decryption keys.

An in-browser system designed to offer No-App-Changes, No-Reuse and No-Phishing is the Implemented, Open and deployed PwdHash [33]: when the user enters a web password, PwdHash replaces it with a hash of the original password and domain name. Even if the same password is used at several sites, each site sees a different version; in particular, the passwords seen by the phishing and phished site are different. Weak passwords are still vulnerable to dictionary attacks but the benefits above are provided at Quasi-No-Cost.

[78] LPWA [19], an early (1997) Implemented "proxy, pseudo" SSO, despite being "privacy-invading" by our definition because the server was under the control of Lucent, offered an insightful discussion of web privacy and introduced a variety of useful mechanisms to protect it (auto-generated aliases, per-site email addresses etc).

[79] Though note that the token itself had to be activated once a day with a PIN.

An important precursor of the Pico is the "Phoolproof Phishing Protection" system by Parno, Cuo and Perrig [27]. Their objective is not to eliminate passwords (which they still use) but to prevent phishing. Yet their system, whose archetypal function is to protect online banking accounts, introduces many architectural features that Pico also adopts: it involves a cellphone interacting via Bluetooth with the local computer, a web browser plugin to talk to the radio and a full SSL interaction, with mutual authentication, with the remote web server. This Implemented system offers No-Phishing, No-Eavesdropping, No-Keylogging and From-Anywhere. Contrary to Pico, Phoolproof is much closer to a drop-in replacement: it requires only minor modifications to web apps, which already have SSL, and users already have suitable cellphones; it thus arguably offers Quasi-No-Cost, Quasi-No-App-Changes, Quasi-No-Cli-Changes. By design, it does not offer No-Search, because the authors want the user to select the app from a "secure bookmark" on the trusted device[80]. Pico takes a different approach (thus providing Scalable and No-Search) because the "secure bookmark" strategy only works when the request to supply the credentials can be initiated from the token; this would not usually be the case for non-web apps (e.g. a desktop application requests the root password because it needs system privileges to run the function you invoked).

The specific problem of online banking authentication has attracted anti-phishing solutions from industry and academia. The system Implemented and sold by Cronto[81] to banks to protect online transactions is not intended as a password replacement but it is perhaps the first commercial system to use a visual code and a multi-channel protocol to protect against phishing. Johnson and Moore [18] Implemented an anti-phishing device based on a USB dongle with display and buttons, robust against malware having compromised the local computer. The device was also designed to provide audit logs suitable for hostile cross-examination in case of dispute.

The Yubico[82] Yubikey (Implemented) is a very low-cost USB token without moving parts, with one capacitive "button" and no display[83]; at every button press it "types" a one-time password which, in conjunction with a password typed by the user, implements a two-factor authentication system. No changes are required at the prover side (No-Cli-Changes), since the Yubikey just looks like a keyboard, but the verifier side needs to know the next OTP to expect. In order not to require the user to carry one Yubikey per app, the apps may outsource their verification procedure to a common authentication server run by one of them or by a third party (possibly Yubico), thus implementing a kind of

[80] One might argue—we do, and so do Laurie and Singer [22] cited further down—about the wisdom of rating the smartphone a trustworthy device; but, in fairness, Parno et al. have Implemented a working prototype before publishing their paper, which can't be said of the Pico or the Neb. The first prototype of the Pico might one day be implemented on a smartphone too, before moving to more secure hardware.

[81] `http://www.cronto.com`.

[82] `http://www.yubico.com`.

[83] The sCrib (`http://www.smartarchitects.co.uk/`) is similar: it types out passwords while offering No-Cli-Changes and No-App-Changes.

"proxy, true, privacy-invading" SSO. This easy to use and Quasi-Memoryless system offers Quasi-Works-For-All[84], From-Anywhere, No-Search, No-Reuse, No-Surfing and No-Keylogging[85] but is still vulnerable to phishing.

The market-leading RSA SecurID[86] (Implemented, Widely-Used), a small device whose display offers a new "pseudo one-time" password every minute or so, is meant to strengthen authentication using two factors, rather than replacing passwords or improving usability. The two factors together, though they cannot prevent phishing, offer at least No-Weak, No-Reuse and No-Keylogging. But the devastating March 2011 compromise of RSA's database, which forced the recall and reissue of 40 million tokens[87], highlights the dangers of a system without No-TTP.

In an entertaining position paper, Laurie and Singer [22] describe yet another security token. Since a general-purpose OS cannot be secured, they offer a *trusted path* to the user via a stripped-down, single-purpose device (the Neb) that can be made sufficiently simple as to be trustworthy. The Neb is not merely a password replacement: details are sketchy but a crucial part of the proposal appears to be trusted-path validation of the operations performed on the general purpose system (e.g. don't just authenticate user Neo to the blogging site, but display his edits on the Neb to get him to approve them). The Neb would certainly provide at least From-Anywhere, No-Phishing and No-Keylogging, but it is unclear how user Neo authenticates himself to his Neb, casting doubts over it being Loss-Resistant and Theft-Resistant. Indeed, no solution is offered for loss of the device other than "centralized revocation and reissue".

On a different note we cannot fail to mention biometrics, which are certainly Memoryless and (beating Pico) No-Carry; they are also potentially Scalable but perhaps not entirely Secure, Loss-Resistant or Theft-Resistant. They offer all the usability benefits of Pico except Continuous[88] but, as for security, they fail to offer No-Phishing, No-Reuse and especially No-Linkage. For privacy, this last point is quite serious: if apps collude, all your authentication acts can be tracked across them, without plausible deniability; and you can't revoke your biometrics and get new ones to regain some privacy.

Concerning Picosiblings, as we mentioned in section 4.1, we first wrote up the idea in 2000 as "family feelings for the Resurrecting Duckling" [36], based on a suggestion from Markus Kuhn. In 2001, Desmedt, Burmester, Safavi-Naini and Wang [7] independently proposed a similar system that featured a threshold scheme. The issue of resharing a shared secret (e.g., for Pico, to change the set of Picosiblings) had been discussed by various authors in other contexts [8,42] but, with specific reference to a quorum of ubicomp devices, Peeters, Kohlweiss and

[84] Like any "true" SSO, it can work with any app whose back-end can be connected to the ASP, and thus it is not limited to web apps.

[85] The password typed by the user can be keylogged but the OTP ensures that, without the token, this captured password is insufficient on its own.

[86] `http://www.rsa.com/node.aspx?id=1156`.

[87] `http://arstechnica.com/security/news/2011/06/rsa-finally-comes-clean-securid-is-compromised.ars`.

[88] Though you might imagine a camera that...

Preneel [31] devised a protocol to *authorize* resharing (in order to prevent an attacker from breaking the scheme by altering the set of devices holding shares) and, with Sulmon [32], also investigated its usability.

In closing we note that several other approaches to the problems of passwords we highlighted in the introduction still ask users to remember something. We won't discuss the many proposals that involve remembering secrets other than passwords (images, gestures, paths ...), as they still all go against what we initially listed as our primary requirement, namely a MEMORYLESS solution[89]; but there are still a few papers we wish to mention.

Jakobsson and Akavipat [17] IMPLEMENTED Fastwords, which are passphrases with special properties, made of several dictionary words. The authors argue that Fastwords are easier to remember, easier to type (especially on smartphones, thanks to auto-completion) and yet of higher entropy than the average password. Like all solutions that involve memorizing and then retyping secrets, Fastwords can't offer MEMORYLESS, SCALABLE, NO-PHISHING, NO-TYPING or NO-KEYLOGGING. They do, however, work with existing smartphones (NO-CLI-CHANGES) and thus, in the context of web apps, can be deployed unilaterally by upgrading only the servers: a major advantage that breaks the vicious circle faced instead by Pico.

A different viewpoint is that passwords are still OK and that it's just the password rules that are unreasonable. In a provocative and stimulating paper, Herley and van Oorschot [16] observe that no other technology offers a comparable combination of cost, immediacy and convenience and that the often-repeated statement that passwords are dead has proved "spectacularly incorrect". The well-argued thesis that Herley, Florêncio et al. develop over several papers [12,15,11] is that we should alleviate the users' memory burden by allowing them to use simpler passwords—because policies requiring stronger passwords impose a definite usability cost without returning a matching security improvement. While we agree that password policies are unreasonably complex, we are not convinced that allowing users to adopt less complex passwords will be sufficient to solve, rather than just temporarily mitigate, the usability problems of passwords, especially if the number of necessary passwords continues to grow. We'd much prefer a MEMORYLESS solution potentially SCALABLE to thousands of credentials. Besides, remembering the passwords is not the only usability problem—that's why we also sought NO-TYPING. And, besides usability, as these authors are first to acknowledge, passwords still suffer from phishing and keylogging.

[89] We do not believe that any of the "remember this other thing instead of a password" methods can possibly offer a long term solution to the fact that users must now remember many secrets, all different, all unguessable, and change them every so often. Systems based on human memory can't scale to hundreds of accounts unless they turn into SSO variants where the user effectively only remembers one secret (and then it's the SSO that's really solving the problem, not the "remember a non-password" method). And that's before considering their sometimes dubious claims about usability and memorability, and their susceptibility to smart guessing attacks that anticipate the way in which users choose their memorable secrets.

7 Conclusions

It is no longer reasonable for computer security people to impose on users such a self-contradictory and unsatisfiable set of password management requirements. Besides, there are too many passwords per user, and it's only going to get worse.

Pico is perhaps the first proposal of an alternative that would eliminate passwords altogether, not just for web sites. It is presented as a clean-slate solution, without regard for backwards compatibility, to explore not the relatively crowded design space of "what can we do to ease today's password pain a little?" but the more speculative one of "what's the best we could do if we didn't have the excuse that we have to support existing systems?". You may use Pico as a springboard to explore the question: "if this isn't the best alternative, why isn't it, and how could we do better?".

There is no expectation that Pico will replace passwords overnight. Even after Pico is debugged security-wise and is ready for prime time usability-wise[90], the success of its deployment will crucially depend on its ability to reach a critical mass of users, capable of self-sustaining a chain reaction of growth[91].

Many password replacement solutions trade off some security to offer greater usability—or vice versa. With Pico, instead, we move to a different curve: more usability *and* more security at the same time. It's not a trade-off, it's a trade-up! It will be hard to break the compatibility shackles, but non-geek users deserve a better deal than we have been offering them until now with passwords[92].

To encourage and facilitate widespread adoption, I have decided not to patent any aspect of the Pico design. If you wish to improve it, build it, sell it, get rich and so forth, be my guest. No royalties due. Just give credit by citing this paper.

Acknowledgements

I am indebted and grateful to Virgil Gligor and Adrian Perrig for their generous and intellectually stimulating hospitality, as the first draft of this paper (repeatedly rewritten since then) originated while I was on sabbatical at Carnegie Mellon's Cylab in 2010. I am grateful to Markus Jakobsson, Joe Bonneau, Attilio Stajano, Omar Choudary, Angela Sasse, Ben Laurie, Matt Blaze, Alf Zugenmaier, "Handslive", "Richard", Paul van Oorschot, Norbert Schmitz, Eirik Schwenke, James Nobis, Per Thorsheim, Erlend Dyrnes, Kirsi Helkala, Cormac

[90] One of the most interesting questions, to be answered through user studies with real prototypes, will be whether managing a family of Picosiblings is felt to be less complicated than managing a collection of passwords.

[91] As Per Thorsheim commented at Passwords^11: "Just get Facebook to adopt it!".

[92] We reiterate what we said in the introduction: passwords may be seen as cheap, easy and convenient for those who deploy them, but they are not for those who have to use them (subject to mutually incompatible constraints) and who have no way of pushing back. App providers may just think "Why should my password be a problem if users are already managing passwords for every other app?", but that's just a *tragedy of the commons*.

Herley, Virgil Gligor, Adrian Perrig, Paul Syverson, Sid Stamm, Andreas Pashalidis, Mark Corner, Bart Preneel, Dirk Balfanz and Carl Landwehr for insightful comments and additional references, as well as to all the other attendees of the Security Protocols Workshop 2011 in Cambridge, UK (whose comments appear in the proceedings), those of the ISSA conference 2011 in Dublin, Ireland, of the Passwords^11 conference in Bergen, Norway, of the Security and Human Behaviour 2011 workshop in Pittsburgh, PA, USA, of the Usenix Security 2011 conference in San Francisco, CA, USA and of the IEEE RTCSA 2011 conference in Toyama, Japan for providing stimulating feedback and probing questions as I presented an evolving version of this work.

References

1. Anne Adams and M. Angela Sasse. "Users are not the enemy". *Communications of the ACM*, **42**(12):40–46, December 1999. `http://hornbeam.cs.ucl.ac.uk/hcs/people/documents/Angela%20Publications/1999/p40-adams.pdf`.
2. Ross J. Anderson and Mike Bond. "The Man-in-the-Middle Defence". In B. Christianson et al. (ed.), "Proc. Security Protocols Workshop 2006", vol. 5087 of *LNCS*, pp. 153–156. Springer, 2009. `http://www.cl.cam.ac.uk/~mkb23/research/Man-in-the-Middle-Defence.pdf`.
3. Adam Beautement, M. Angela Sasse and Mike Wonham. "The compliance budget: managing security behaviour in organisations". In "Proc. New Security Paradigms Workshop 2008", pp. 47–58. ACM, 2008. `http://hornbeam.cs.ucl.ac.uk/hcs/people/documents/Adam%27s%20Publications/Compliance%20Budget%20final.pdf`.
4. Joseph Bonneau and Sören Preibusch. "The password thicket: technical and market failures in human authentication on the web". In "Proc. 9th Workshop on the Economics of Information Security", Jun 2010. `http://preibusch.de/publications/Bonneau_Preibusch__password_thicket.pdf`.
5. Omar Choudary. *The Smart Card Detective: a hand-held EMV interceptor*. Master's thesis, University of Cambridge, 2010. `http://www.cl.cam.ac.uk/~osc22/docs/mphil_acs_osc22.pdf`.
6. Mark D. Corner and Brian D. Noble. "Zero-interaction authentication". In "Proc. ACM MobiCom 2002", pp. 1–11. 2002. `http://www.sigmobile.org/awards/mobicom2002-student.pdf`.
7. Y. Desmedt, M. Burmester, R. Safavi-Naini and H. Wang. "Threshold Things That Think (T4): Security Requirements to Cope with Theft of Handheld/Handless Internet Devices". In "Proc. Symposium on Requirements Engineering for Information Security", 2001.
8. Yvo Desmedt and Sushil Jajodia. "Redistributing Secret Shares to New Access Structures and Its Applications". Tech. Rep. ISSE-TR-97-01, George Mason University, Jul 1997. `ftp://isse.gmu.edu/pub/techrep/9701jajodia.ps.gz`.
9. Saar Drimer and Steven J. Murdoch. "Keep your enemies close: distance bounding against smartcard relay attacks". In "Proc. USENIX Security Symposium", pp. 87–102. Aug 2007. `http://www.cl.cam.ac.uk/~sd410/papers/sc_relay.pdf`.
10. Dinei Florêncio and Cormac Herley. "One-Time Password Access to Any Server without Changing the Server". In "Proc. 11th Information Security Conference", pp. 401–420. Springer-Verlag, Berlin, Heidelberg, 2008. `http://research.microsoft.com/~cormac/Papers/otpaccessanyserver.pdf`.

11. Dinei Florêncio and Cormac Herley. "Where do security policies come from?" In "Proc. SOUPS 2010", pp. 10:1–10:14. ACM, 2010. `http://research.microsoft.com/pubs/132623/WhereDoSecurityPoliciesComeFrom.pdf`.
12. Dinei Florêncio, Cormac Herley and Baris Coskun. "Do strong web passwords accomplish anything?" In "Proc. USENIX HOTSEC 2007", pp. 10:1–10:6. 2007. `http://research.microsoft.com/pubs/74162/hotsec07.pdf`.
13. Gerhard P. Hancke and Markus G. Kuhn. "An RFID Distance Bounding Protocol". In "Proc. IEEE SECURECOMM 2005", pp. 67–73. 2005. `http://www.cl.cam.ac.uk/~mgk25/sc2005-distance.pdf`.
14. Feng Hao, Ross Anderson and John Daugman. "Combining Crypto with Biometrics Effectively". *IEEE Transactions on Computers*, **55**(9):1081–1088, September 2006. `http://sites.google.com/site/haofeng662/biocrypt_TC.pdf`.
15. Cormac Herley. "So Long, and No Thanks for the Externalities: the Rational Rejection of Security Advice by Users". In "Proc. New Security Paradigms Workshop 2009", ACM, 2009. `http://research.microsoft.com/users/cormac/papers/2009/SoLongAndNoThanks.pdf`.
16. Cormac Herley and Paul C. van Oorschot. "A Research Agenda Acknowledging the Persistence of Passwords", 2011. In submission.
17. Markus Jakobsson and Ruj Akavipat. "Rethinking Passwords to Adapt to Constrained Keyboards", 2011. `http://www.markus-jakobsson.com/fastwords.pdf`. In submission.
18. Matthew Johnson and Simon Moore. "A New Approach to E-Banking". In Ú. Erlingsson et al. (ed.), "Proc. 12th Nordic Workshop on Secure IT Systems (NORDSEC 2007)", pp. 127–138. Oct 2007. `http://www.matthew.ath.cx/publications/2007-Johnson-ebanking.pdf`.
19. David M. Kristol, Eran Gabber, Phillip B. Gibbons, Yossi Matias and Alain Mayer. "Design and implementation of the Lucent Personalized Web Assistant (LPWA)". Tech. rep., Bell Labs, 1998.
20. C. E. Landwehr. "Protecting unattended computers without software". In "Proceedings of the 13th Annual Computer Security Applications Conference", pp. 274–283. IEEE Computer Society, Washington, DC, USA, Dec 1997. ISBN O-8186-8274-4. `http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA465472`.
21. Carl E. Landwehr and Daniel L. Latham. "Secure Identification System", 1999. US Patent 5,892,901, filed 1997-06-10, granted 1999-04-06.
22. Ben Laurie and Abe Singer. "Choose the red pill and the blue pill: a position paper". In "Proc. New Security Paradigms Workshop 2008", pp. 127–133. ACM, 2008. `http://www.links.org/files/nspw36.pdf`.
23. Tsutomu Matsumoto, Hiroyuki Matsumoto, Koji Yamada and Satoshi Hoshino. "Impact of Artificial Gummy Fingers on Fingerprint Systems". In "Proc. SPIE, Optical Security and Counterfeit Deterrence Techniques IV", vol. 4677. 2002. `http://cryptome.org/gummy.htm`.
24. Jonathan M. McCune, Adrian Perrig and Michael K. Reiter. "Seeing-Is-Believing: Using Camera Phones for Human-Verifiable Authentication". In "Proc. IEEE Symposium on Security and Privacy 2005", pp. 110–124. `http://sparrow.ece.cmu.edu/group/pub/mccunej_believing.pdf`. Updated version in *Int. J. Security and Networks* **4**(1–2):43–56 (2009) at `http://sparrow.ece.cmu.edu/group/pub/mccunej_ijsn4_1-2_2009.pdf`.
25. Anthony Nicholson, Mark D. Corner and Brian D. Noble. "Mobile Device Security using Transient Authentication". *IEEE Transactions on Mobile Computing*,

5(11):1489–1502, Nov 2006. http://prisms.cs.umass.edu/mcorner/papers/tmc_2005.pdf.

26. Donald A. Norman. *The Psychology of Everyday Things*. Basic Books, 1988. ISBN 0-385-26774-6. Also published as *The Design of Everyday Things* (paperback).
27. Bryan Parno, Cynthia Kuo and Adrian Perrig. "Phoolproof Phishing Prevention". In G. Di Crescenzo et al. (ed.), "Proc. Financial Cryptography 2006", vol. 4107 of *LNCS*, pp. 1–19. Springer, 2006. http://sparrow.ece.cmu.edu/group/pub/parno_kuo_perrig_phoolproof.pdf.
28. Andreas Pashalidis. "Accessing Password-Protected Resources without the Password". In M. Burgin et al. (ed.), "Proc. CSIE 2009", pp. 66–70. IEEE Computer Society, 2009. http://kyps.net/xrtc/cv/kyps.pdf.
29. Andreas Pashalidis and Chris J. Mitchell. "A taxonomy of single sign-on systems". In R. Safavi-Naini et al. (ed.), "Proc. 8th Australasian conference on Information security and privacy", vol. 2727 of *LNCS*, pp. 249–264. Springer, 2003. http://www.isg.rhul.ac.uk/cjm/atosso.pdf.
30. Andreas Pashalidis and Chris J. Mitchell. "Impostor: a single sign-on system for use from untrusted devices". In "Proc. IEEE GLOBECOM 2004", vol. 4, pp. 2191–2195. 2004. http://www.isg.rhul.ac.uk/cjm/iassos2.pdf.
31. Roel Peeters, Markulf Kohlweiss and Bart Preneel. "Threshold Things That Think: Authorisation for Resharing". In Jan Camenisch and D. Kesdogan (eds.), "Proceedings of iNetSec 2009 – Open Research Problems in Network Security", vol. 309 of *IFIP Advances in Information and Communication Technology*, pp. 111–124. Zurich,CH, 2009. http://www.cosic.esat.kuleuven.be/publications/article-1223.pdf.
32. Roel Peeters, Markulf Kohlweiss, Bart Preneel and Nicky Sulmon. "Threshold things that think: usable authorization for resharing". In "Proceedings of the 5th Symposium on Usable Privacy and Security", SOUPS '09, pp. 18:1–18:1. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-736-3. http://cups.cs.cmu.edu/soups/2009/posters/p1-peeters.pdf.
33. Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh and John C. Mitchell. "Stronger Password Authentication Using Browser Extensions." In "Proc. Usenix Security", pp. 17–32. 2005. http://crypto.stanford.edu/PwdHash/pwdhash.pdf.
34. Stuart Schechter, Serge Egelman and Robert W. Reeder. "It's not what you know, but who you know: a social approach to last-resort authentication". In "Proc. CHI 2009", pp. 1983–1992. http://research.microsoft.com/pubs/79349/paper1459-schechter.pdf.
35. Adi Shamir. "How to Share a Secret". *Communications of the ACM*, **22**(11):612–613, Nov 1979. http://securespeech.cs.cmu.edu/reports/shamirturing.pdf.
36. Frank Stajano. "The Resurrecting Duckling—What Next?" In B. Christianson et al. (ed.), "Proc. Security Protocols Workshop 2000", vol. 2133 of *LNCS*, pp. 204–214. Springer, 2001. http://www.cl.cam.ac.uk/~fms27/papers/2000-Stajano-duckling.pdf.
37. Frank Stajano. *Security for Ubiquitous Computing*. Wiley, 2002. ISBN 0-470-84493-0. Contains the most complete treatment of the Resurrecting Duckling [38].
38. Frank Stajano and Ross Anderson. "The Resurrecting Duckling: Security Issues in Ad-Hoc Wireless Networks". In B. Christianson et al. (ed.), "Proc. Security Protocols Workshop 1999", vol. 1796 of *LNCS*, pp. 172–182. Springer, 2000. http://www.cl.cam.ac.uk/~fms27/papers/1999-StajanoAnd-duckling.pdf.
39. Frank Stajano, Ford-Long Wong and Bruce Christianson. "Multichannel protocols to prevent relay attacks". In R. Sion (ed.), "Proc. Financial Cryptography 2010",

vol. 6052 of *LNCS*, pp. 4–9. Springer, 2010. http://www.cl.cam.ac.uk/~fms27/papers/2009-StajanoWonChr-relay.pdf.

40. Roy Want and Andy Hopper. "Active Badges and Personal Interactive Computing Objects". *IEEE Transactions on Consumer Electronics*, **38**(1):10–20, Feb 1992. http://nano.xerox.com/want/papers/pico-itce92.pdf.
41. Ford-Long Wong and Frank Stajano. "Multi-channel Protocols". In Christianson et al. (ed.), "Proc. Security Protocols Workshop 2005", vol. 4631 of *LNCS*, pp. 112–127. Springer-Verlag. http://www.cl.cam.ac.uk/~fms27/papers/2005-WongSta-multichannel.pdf. Updated version in *IEEE Pervasive Computing* **6**(4):31–39 (2007) at http://www.cl.cam.ac.uk/~fms27/papers/2007-WongSta-multichannel.pdf.
42. Theodore M. Wong, Chenxi Wang and Jeannette M. Wing. "Verifiable Secret Redistribution for Archive System." In "IEEE Security in Storage Workshop'02", pp. 94–105. 2002. http://www.cs.cmu.edu/~wing/publications/Wong-Winga02.pdf.