Better authentication: password revolution by evolution

Daniel R. Thomas and Alastair R. Beresford

Computer Laboratory, University of Cambridge, United Kingdom
`firstname.lastname@cl.cam.ac.uk`

Abstract. We explore the extent to which we can address three issues with passwords today: the weakness of user-chosen passwords, reuse of passwords across security domains, and the revocation of credentials. We do so while restricting ourselves to changing the password verification function on the server, introducing the use of existing key-servers, and providing users with a password management tool. Our aim is to improve the security and revocation of authentication actions with devices and end-points, while minimising changes which reduce ease of use and ease of deployment. We achieve this using one time tokens derived using public-key cryptography and propose two protocols for use with and without an online rendezvous point.

Keywords: Authentication, Public-key cryptography, passwords, one time token

1 Introduction

Text-based passwords, passphrases or PINs are the dominant method of authenticating users today. They are used to establish the identity of an individual to an *end-point*, by which we mean either: a local physical device controlled by the user, or software running locally on it (local end-point); or a service available over a network (remote end-point). With the rise of the Internet, and particularly the Web, the number of remote end-points has risen dramatically with the average user managing 25 web accounts in 2007 [13]. Similarly, as ubiquitous computing takes hold, the number of devices individuals carry or use has increased along with the number of local end-points. Rather than sharing access to one home PC, many users now have multiple desktops as well as a laptop, tablet and smartphone. These all need lots of credentials – shared secrets – not just to access apps on the device or remote services but networks too. For example Figure 1 shows the distribution of non-open WiFi credentials we found on 8622 devices in the Device Analyzer project. Many users have more than 5 sets of wireless credentials and some have more than 40.

Passwords or PINs are popular because they are easy to use (incorrectly) and easy to deploy [7]. They are also popular because they are the incumbent technology: application developers believe they know how to deploy password authentication securely, and users know what a password is and do not need

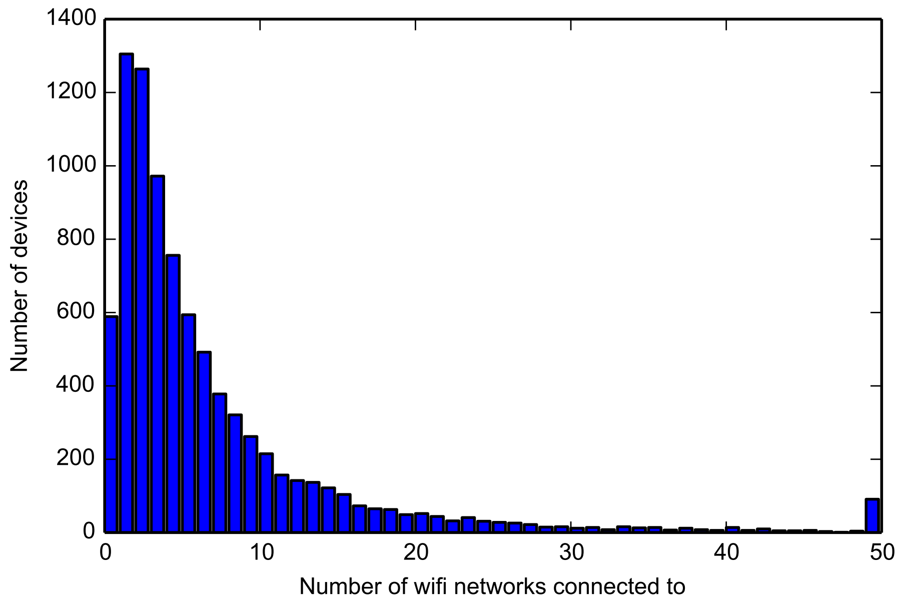

Fig. 1. Number of WiFi credentials (WEP, WPA etc.) used while Device Analyzer [32] was installed on 8622 devices (only counting if they participated for at least a week) devices which never connected to any wireless networks not counted. Data collected between 2011 and 2014.

any new hardware as almost all devices requiring authentication today have a (soft) keyboard to enter numeric or textual data. It has also been speculated that authentication by passwords is popular because it conveys a sense of exclusivity and membership, and is therefore attractive to businesses which otherwise do not need to authenticate individuals [6].

We cannot remember strong passwords without strong incentives and good technical understanding [1]; attributes which the vast majority of users (quite reasonably) lack. As a result, the use of passwords for authentication has always been problematic because individuals choose weak passwords and reuse passwords across security domains. These two weaknesses mean that brute-force attacks are possible, and the compromise of one end-point or device compromises another end-point or device. More recently, as the number of passwords and devices has increased, the revocation of passwords in the event of loss is becoming both increasing difficult and increasingly important. Since passwords represent a shared secret, the loss of a device means that many passwords across many end-points and devices need to be reset. Can you remember all the passwords you would need to reset if you lost your smartphone? What would be the effect on your access to your remaining devices and end-points if you did so?

Whilst the use of passwords for authentication has remained largely unchanged for 40 years, there have been two important changes which have been widely adopted and deployed. Firstly, on the end-point and device side, different password verification functions have been used from DES Crypt [21] through MD5 to SHA512 and scrypt [24]. Secondly, on the user side, the automation of password management, in the form of keychains, browser features and general password managers, has become popular. In a survey of password behaviour [26], 28% of respondents said they always use a "remember my password" function, and 70% have made use of such a feature. Unfortunately popular password managers do not address important security issues: they allow users to input weak passwords which can also be shared across multiple domains (though many offer the facility to generate them), and they do not address the issue of revocation. Some also do not deal with backup and frequently do not encrypt the password database making it vulnerable to theft (for example if the user does not select a master password).

In this paper we explore the extent to which we can address three issues with passwords today: the weakness of user-chosen passwords, reuse of passwords across security domains, and the revocation of credentials. We do so while restricting ourselves to (i) changing the password verification function on the server, (ii) introducing the use of existing key-servers, and (iii) providing users with a password management tool. Our aim is to improve the security and revocation of authentication actions with devices and end-points, while minimising changes which reduce ease of use and ease of deployment and so make this a practical system.

2 Design overview

Our approach is similar to Monkeysphere [§ 4.1] in that we intend to replace password-based shared secrets with public-key cryptography. Our design operates in a similar manner to a password management tool, which are widely deployed and familiar to many users. This has the advantage that existing software stacks, which assume the existence of a secret encoded as a string, continue to function provided the underlying verification function is modified.

Our design makes use of three components: key-servers, devices and end-points. Key-servers maintain an append-only log of public-keys and associated fingerprints (hashes), revocation certificates and signatures. Personal devices each have a public/private key pair for every identity and publish their public-keys on key-servers.

Users authenticate themselves to end-points by using a device they control. Authentication between the device and the end-point takes place over an established secure channel (e.g. physical proximity or TLS) and requires a single message generated by the device to be delivered to the end-point (e.g. by typing).

2.1 Threat model

Informally our adversary has the following abilities. Information sent and received from the key-servers can be observed, tampered with or dropped. Information sent over the channel between the device and end-point cannot be observed or tampered with. End-points in multiple security domains can be compromised by the adversary. Under those constraints an adversary should not be able to authenticate to any end-points in domains without compromised end-points. A compromised end-point should only be able to authenticate as the user to other end-points in that domain if the user attempts to authenticate to it (unlike with shared secrets where end-points can pretend to be any user to other end-points).

2.2 Key distribution

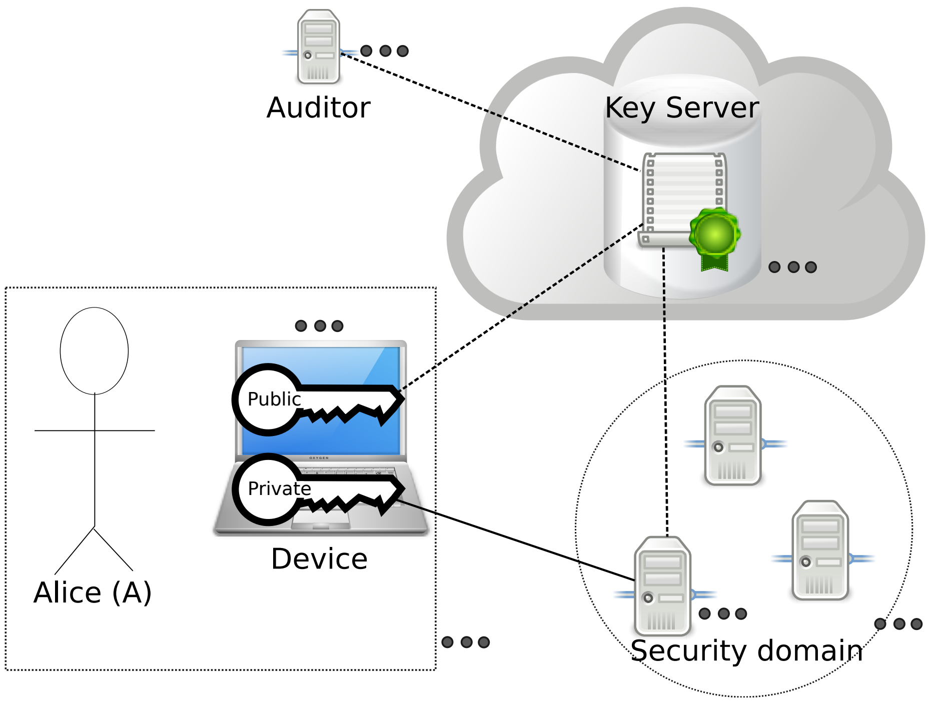

Fig. 2. Each user has a device(s) with a keypair(s). The public-keys, signatures and revocation certificates are published to the key-server(s) which maintain an append-only log, which is audited by the auditor(s). The key fingerprints are registered with the security domain(s) which can then subscribe to the key-server(s) and authenticate users using a one time token.

A diagram showing the different actors in the key distribution is given in Figure 2.

A user is a person who has personal computing devices, such as a smartphone, a tablet, a laptop, a desktop or a smartwatch.

A user may have multiple unlinked identities, perhaps one for their legal identity 'Joe Bloggs' another for when anonymous 'A. N. Other' and another for secret activity 'James Bond'. For each identity on each device the user can generate a public / private key pair to use for authenticating that identity using that device.

These public-keys are distributed to key-servers. Key-servers do not need to be trusted to provide integrity or authenticity as all their actions can be audited as all the data they store is public and signed. For example a Certificate Transparency [19] style system could be used.

Users need to authenticate within many different security domains. At home they need to authenticate to their own systems, with others at work, online they need to authenticate to many mutually untrusting online providers such as identify providers, banks and governments. These security domains have many end-points at which authentication can be required.

Key generation For each identity on each device the user generates a key-pair associated with that identity with a creation timestamp. This metadata is then signed using the private key. A revocation certificate should be generated and stored somewhere safe against future key compromise. The public key and associated metadata are published to the key-servers.

Adding new keys Users need to link the public-keys for the same identity on different devices:

1. Input the fingerprint of the public-key for that identity on device A into device B
2. Request device B sign the public-key input
3. Publish the signature to the key-servers

Then repeat reversing A and B. All the keys with fingerprints signed by one key are treated as being equivalent by the verifying end-point.

Registration At registration for this scheme at an end-point in a security domain:

1. The user provides the key fingerprint for one of their keys for the relevant identity.
2. The end-point requests that public-key and the transitive closure of all the public-keys signed by the corresponding private key as being equivalent keys.
3. The end-point stores the fingerprints of all these public keys as being those supplied at registration by the user after confirming this with the user.
4. The security domain registers with the key-servers to be told about any relevant new signatures or revocations on these keys.

Revocation When a key-pair needs to be revoked then the adversary may have the key-pair and the legitimate user may not have it. If the legitimate user still has the key-pair then they can issue a new revocation certificate specifying the date when the key-pair was compromised. Then any signatures or authentication attempts made using that key-pair after that date are invalid. Otherwise they can publish the revocation certificate generated originally invalidating all signatures made with that key (hence the importance of following the transitive closure of signatures at registration time, this prevents the user losing all access when revoking a key). Additionally a valid uncompromised key-pair can be used to sign a modification to the revocation certificate specifying a later date when this should apply, however this key must still be valid even in the case of the original revocation date so that an adversary cannot publish a modification which moves the revocation date later after compromising the key.

Since the public-keys and signatures are published to an auditable append-only log with timestamps, the adversary who has compromised a key cannot produce signatures with dates in the past as they will only appear in the log after their real creation date.

Thus an adversary can only push the revocation time earlier, invalidating valid signatures and so while they could perform a denial of service, they could not authenticate as the user. They also cannot cause key-pairs which they have not compromised and which were valid at registration to be invalidated.

2.3 Simple one time token authentication protocol (SOTTA)

We assume that the end-point is already authenticated to the user, e.g. by physical proximity or Transport Layer Security (TLS) (for all the flaws of the CA hierarchy [9]). We also assume that they are communicating over a secure channel resistant to man in the middle (MITM) attacks.

To ensure that tokens valid for one domain are not valid for use at other domains we include the domain's name (such as a DNS domain name) D in the token. To ensure that tokens are fresh without having specific shared state we incorporate a timestamp.

Notation (in the style of the BAN logic [8]):

- K_X is the public-key associated with the identity X and K_X^{-1} is the private key for X.
- I is the time interval in which a token is valid.
- t is the current epoch time on Alice's (A) device's clock, t' is end-point's (server, S) time.
- The floor function $\lfloor t \rfloor$ quantises a time to the system's quantum Q (e.g. one minute).
- The $||$ operator concatenates of the octets of its arguments each preceded by its own length represented as 4 bytes, bigendian.
- $\{y\}_{K_X^{-1}}$ denotes the signature of y with X's private key.

The initial state for authentication is:

- Alice (A) has a device with K_A, K_A^{-1} and a clock synchronised to within Δt of UTC.
- The end-point (server, S) in domain with name D has K_A and a clock within $\Delta t'$ of UTC.
- $|\Delta t - \Delta t'| + \text{transmissiondelay} < I$
- A has a secure connection (provides confidentiality and integrity) to S which authenticates S

Then A can authenticate to S in one step by sending a token:

$$A \rightarrow S : A, \{D || \lfloor t \rfloor\}_{K_A^{-1}} \tag{1}$$

S knows D and assumes its time t' is within I of t (even after the transmission delay) and has received the one time token o from A. It can then check whether:

$$\exists t_c . t' - \frac{I}{2} \leq t_c \leq t' + \frac{I}{2} \bigwedge t_c \bmod Q = 0 \bigwedge \text{verify}(D || t_c, o, K_A) \tag{2}$$

That is, is there a candidate time t_c within $\frac{I}{2}$ of S's clock t' which causes the one time token o to verify. If I is 5 system quanta then S must perform at most 5 verifications before rejecting an authentication attempt. Hopefully usually only one verification would be required if S picks the order to verify t_c's in carefully (most likely first). It does not require much storage for S to check that it has not seen this token before, since tokens expire quickly.

While in many circumstances a password-manager style application could be used to input o, in some cases the user must still type the password. Hence the size $|\{D || \lfloor t \rfloor\}_{K_A^{-1}}|$ is important as this may have to be typed in by the user. This size is determined by the signature algorithm used.

2.4 Signature algorithm

User text entry of random strings is a low-bandwidth channel with a high error rate and so we want the signature size to be as small as possible. Table 1 gives the number of characters required for different bit lengths with different encodings. With symmetric key-based signatures a client can provide a truncated signature to a server and the server can still verify it by computing the signature and truncating it. With public-key cryptography the server cannot generate a signature, only verify signatures and so the client must send the full signature. For 128 security bits a 3072 bit RSA [27] key is required [2] and even for 80 security-bits a 1024 bit RSA key is required (and that could be brute forced). With RSA the signature length is the same as the key length and 1024 bits takes 172 Alphanumeric[1] characters to represent, which a user will not be willing to type in. DSA[2] (and ECDSA) [12] need signature lengths four times the security-bits size and so need 320 bits for 80 security-bits or 512 bits for 128 security-bits which takes 54 and 86 Alphanumeric characters respectively.

Table 1. Encoding sizes for different bit lengths

Bits	Bytes	numeric	alphabetical	Alphanumeric	Algorithm
		[0-9]	[a-z]	[A-Za-z0-9]	
32	4	10	7	6	
64	8	20	14	11	
80	10	25	18	14	
128	16	39	28	22	
160	20	49	35	27	
256	32	78	55	43	Minimum
320	40	97	69	54	BSL?
512	64	155	109	86	DSA
1024	128	309	218	172	
3072	384	925	654	516	RSA

Table 2. Bits required for different signature schemes for different numbers of security-bits

Scheme	80	112	**128**	256
RSA	1024	2048	3072	15360
DSA	320	112	512	1024
BSL	171 [5], 160 [3]	224 [3]	256 [3]	640 [17]
Minimum	160	224	256	512

There are shortened signature schemes for DSA [22, 25, 23] but they seem to use message recovery which does not help as the message is implicit and so only the signature needs transmission. We are also suspicious of such schemes as the DSS clearly states that the ephemeral key k used in the signing process must be cryptographically random and secret [12], indeed leaking any bits of it progressively compromises the private key [15].

The BSL signature scheme [5] provides signature lengths approaching the theoretical minimum of twice the number of security-bits but is not widely deployed. The theoretical limit for a hash function is double the number of security-bits because of the birthday problem. A comparison of the signature lengths of different schemes is given in Table 2.

There are other schemes: NTRUSign was broken in 2012 [11]. Lamport signatures [18] could be used but would need a different scheme with one public-key chain per domain which would make scalability harder. Schnoor signatures [29] have similar properties to DSA but are less widely used.

Ideally we would like a public-key signing scheme with a signature size providing at least 112 security-bits[3] using fewer than 256 bits in total (43 Alphanumeric characters) but there is no widely used scheme with this property – BSL might

[1] [A-Za-z0-9]

[2] DSA is broken if the random number used for nonces is biased which is problematic as frequently devices have bad random number generators that would leak the private key [15].

[3] NIST minimum number of security-bits to 2030 [2].

be a suitable candidate. The user can type the signature in but we can also input it either by automatically inputting it [§2.5] or by using online assistance [§2.6].

2.5 Automatic entry (AOTTA)

If the device containing the keys is the same device as the one where the token needs to be input (e.g. if the keys are on a smartphone and authentication to a website through the mobile browser is being attempted) then the token can simply be copied – as with passwords and password-managers. If the device containing the keys emulates a keyboard (e.g. using Bluetooth) to the machine into which the token needs to be input, then when the user pushes a button on the device the token can be automatically typed. If the device containing the keys has a screen and the device where the token needs to be input has a camera then a QR code could be used. Alternatively audio networking could be used if the device has a speaker and the end-point has a microphone [20].

2.6 Online assisted one time token protocol (OOTTA)

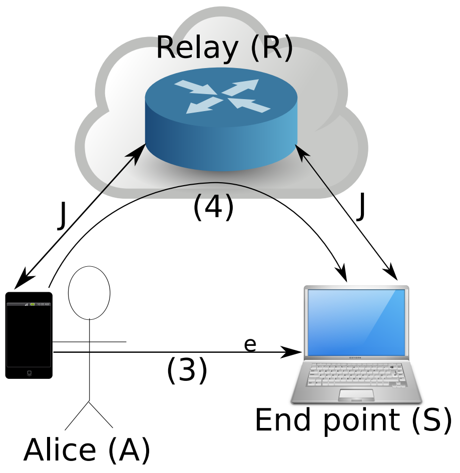

Fig. 3. Architecture of OOTTA: the existing channel is labelled 'e' and communication via the relay is set up using J-PAKE 'J'

When both the device with the keys and the end-point are online then they can rendezvous and authenticate with a short token s transmitted locally, this is shown in Figure 3. Data about a device's preferred rendezvous point can be associated with the public-key and so does not need to be input. All that needs to be input is the identity of the user A and a random token s which can be low entropy. Then Password Authenticated Key Exchange (PAKE) protocols

[§4.2] such as J-PAKE [14] or SPEKE [16] can be used to generate a shared key between the device with the keys and the end-point. Since this shared key is authenticated by s the device with the keys can perform the simple one time token authentication protocol (SOTTA) [§2.3] over a connection encrypted and authenticated with the shared key with the same result as before.

To analyse the latency and steps in this protocol more precisely we will use J-PAKE and assume that messages must be forwarded via a relay (R)[4] as the user device and the end-point may not be able to communicate directly due to network address translation.

First A must send the random short secret to the end-point (S) over the existing channel e (e.g. keyboard).

$$A \underset{e}{\rightarrow} S : s \tag{3}$$

Then concurrently A and S each send a message and a response to each other via the relay according to the J-PAKE protocol [14]. From which they can both compute session key $\mathcal{K}$ and confirm it with another request and response.

At this point A and S share a session key $\mathcal{K}$ which they can use for authenticated encryption (denoted $\mathrm{E}_{\mathcal{K}}(x)$) and which has the same properties as the original secure channel e (such as physical proximity) in terms of integrity, confidentiality and the authenticity of S to A. It remains to authenticate A to S which we can do using the SOTTA protocol encrypted with $\mathcal{K}$.

$$A \rightarrow S : \mathrm{E}_{\mathcal{K}}(\{D||\lfloor t \rfloor\}_{K_A^{-1}}) \tag{4}$$

Since the first two pairs of messages are sent concurrently, the total time spent transmitting messages over the network is five times the time it takes to send a message from A to S via the relay R or 2.5 RTTs which equates to about 1 second in the worst case.[5]

2.7 Deployment

In practice we would expect large technology companies and perhaps internet infrastructure organisations such as ICANN to run key-servers (Google has already indicated a willingness to run a Certificate Transparency server which is a similar service). Relays could be self hosted or run by technology companies interested in increasing lock-in for their users. End-points range from desktops, laptops and other local devices through to servers running websites of all descriptions. Devices containing private-keys could be dedicated hardware, or commodity smartphones, desktops or laptops.

[4] We are going to ignore TCP handshakes here and retransmissions as these are implementation details (we could implement with UDP).

[5] A and S adjacent and R on the opposite side of the world

3 Evaluation

Many different authentication schemes have been proposed to replace passwords. Bonneau et. al. [7] have conducted an exhaustive survey of authentication schemes which we will not repeat here. It proposes the UDS framework criteria for evaluating authentication schemes which we use to evaluate our proposal and defines the terms in *italics* which follow.

In the following we will use OTTA for our proposal in general, SOTTA for the simple manual input case, AOTTA for the automatic entry case and OOTTA for the online assisted case.

Usability: OTTA is *Memorywise-Effortless* as users do not need to remember any secrets. It is *Scalable-for-Users* as it can be used for hundreds or thousands of domains without an additional burden on the user (they might need to input the domain name into their device once per domain but this still scales). It is not *Nothing-to-Carry* as the user must carry a device with keys. It is not *Physically-Effortless* as the user must type something in. It is hard to evaluate *Easy-to-Learn* without a real system to test against. It could be *Efficient-to-Use* for AOTTA and OOTTA. It should be *Infrequent-Errors* for OOTTA and for AOTTA. It is *Easy-Recovery-from-Loss* as multiple working devices with independent keys are supported and so backups can be taken.

Deployability: OTTA is at least as *Accessible* as passwords. It is *Negligible-Cost-per-User* as we assume users will already have a 'suitable' device such as a smartphone or laptop. A dedicated device might provide better security but is not necessary. It is not *Server-Compatible* as some changes would be required at the server end to support it. However it would be compatible with existing users still using passwords as it just requires different treatment of the received token. It is *Browser-Compatible* as the browser only sees a text password input, protocol logic is at the end-point and in the device. It is not *Mature* as it has not been implemented. It is *Non-Proprietary* using known algorithms and protocols not encumbered with patents.

Security: OTTA is *Resilient-to-Physical-Observation* as tokens are only valid once and for a short period. An attacker could observe a token and type it in faster than the user but this would only be valid once. Domains should ensure that they request a new token when an authenticated client attempts to add a new key (as with passwords). It is *Resilient-to-Targeted-Impersonation* as it relies on randomly generated keys. It is *Resilient-to-Throttled-Guessing* and *Resilient-to-Unthrottled-Guessing* as the keys would need to be brute forced and that should require 2^{128} work (depending on the security parameter). It is not *Resilient-to-Internal-Observation* at least not if the device with the keys is not a dedicated device that cannot acquire malware. It is *Resilient-to-Leaks-from-Other-Verifiers* as the verifier has nothing that can be used to impersonate a user to a different verifier. It is *Resilient-to-Phishing* as a phising site can only obtain a one time token which would have to be used in real time (the definition of *Resilient-to-Phishing* excludes this and we assume a secure connection to the end-point which authenticates it). It is not *Resilient-to-Theft* in general as we place no restriction on the devices that could be used. If users use strong security to protect their device such that data on it cannot be accessed then they would have this benefit. There is *No-Trusted-Third-Party* since bad key-servers can only compromise availability and many key-servers can be used safely and audited. It is *Requiring-Explicit-Consent* as a user must select which domain they want to authenticate to and then type something/press a button. It is very much not *Unlinkable* as one (set of) public-key(s) is used for all domains, the tokens themselves cannot be used for linking.

4 Related work

Password managers exist that generate unique passwords for each site, but so far they have not been popular. For example, PwdHash [28] automatically generates secure passwords by hashing a user-supplied master password with the domain name of a website and has been downloaded over 100,000 times since 2009.

If end-points supported multiple passwords then a different password could be specified for every (device, end-point) pair but this would not scale with hundreds of end-points and many devices as each password would have to be manually configured. A single dedicated hardware token such as a Pico [30] could be used but this could still be compromised and hence a revocation and resetting of credentials protocol would be required. Additionally someone would need to pay for this new hardware token. A password manager could be used to store one strong password per security domain (e.g. identity provider, work, home) but then this needs to synchronise the secrets between several devices. The compromise of any one device would result in the passwords needing to be reset for all domains. Secure synchronisation is not trivial though there is work on plausible solutions [31].

There are shared secret schemes with better resistance to brute force attacks and weak choice of passwords such as the one time password scheme used by Google's Authenticator [7, p. IV.I.5] or RSA tokens [7, IV.H.1] – but they have the same scalability problems.

SSH can use public-key cryptography to authenticate users to the server and the public-key can be distributed to many end-points. Monkeysphere solves the difficulty of revocation and distribution of new keys in that situation.

4.1 Monkeysphere

Our original motivation for developing SOOTA was to extend our existing scalable distributed authentication mechanism for our servers, which use Monkeysphere and SSH, for the cases where networking on the VM was broken or physical connection to a server was required. Monkeysphere[6] solves the key distribution and revocation problem for SSH[7] by using GPG / PGP with its key-server

[6] `http://web.monkeysphere.info/`

[7] It also aims to augment/replace the CA hierarchy for TLS but that is not our focus.

infrastructure to perform fast revocation. The user provides the domain administrator with a user id (uid) for their keys and the administrator signs the keys they have verified (e.g. in person) with a key trusted by the domain. Then the machines can automatically fetch all the keys which match the uid and verify them using a domain key. Those which verify can then be used to login. The key pair for SSH is embedded as a GPG subkey in the main GPG key and then SSH public-key authentication proceeds as normal with the public subkeys from the verified GPG keys being added to the relevant `authorized_keys` file.

In our group there are two domain administrators who sign the Monkeysphere enabled GPG keys of users that are then used to authenticate to all our machines.[8]

4.2 Password authenticated key exchange (PAKE)

Passwords are low entropy secrets, and they have no resistance against brute force offline attacks where the number of guesses an attacker has is unlimited. However low entropy secrets are secure against online attacks where an attacker has a limited number of guesses. Unfortunately encryption, decryption and signing expose the key used to perform the cryptographic operation to offline brute force attacks and so high entropy keys are required. Password Authenticated Key Exchange (PAKE) schemes use a low entropy password to authenticate a high entropy key which two parties have generated (e.g. by using Diffie-Hellman (DH) key exchange [10]). A secure PAKE scheme only reveals to the two parties whether they have the same password or not on each run of the protocol. So each party knows how many times the protocol has been run and so how many guesses have been had and can refuse to participate in future runs.

J-PAKE [14] is a PAKE scheme which uses DH key exchange and is the only such scheme with a security proof and no known issues, however it is also relatively new (2011). The first such scheme was EKE [4] (1992) but some flaws have been found in it [14]. SPEKE [16] (1996) appears to be better but still has flaws allowing more than one guess of the password on each run [14]. Both EKE and SPEKE have been patent encumbered which has reduced adoption. Hence we consider only J-PAKE for use in our protocol.

5 Discussion and conclusion

The system proposed in this paper could be implemented and make use of existing infrastructure. As with Monkeysphere we want to perform authentication using a specific GPG subkey, perhaps with some additional associated data (the relay information for OOTA). Then we can perform distribution of the public-keys and associated data in the background using higher bandwidth internet links on a periodic basis. Verification can be performed at registration by supplying a uid (such as '`Daniel Robert Thomas <drt24@cam.ac.uk>`') and a key fingerprint (such as '`5017 A1EC 0B29 08E3 CF64 7CCD 5514 35D5 D749 33D9`').

[8] The source code is available `https://github.com/ucam-cl-dtg/dtg-puppet/`

The supplying of the key fingerprint allows the key to be automatically signed, and subsequent keys can be verified by: (i) signature of an existing key, (ii) inputting the the key id within an existing login session, or by (iii) signature of one of the domain admins. Since all the keys are synchronised via the key-server network, it is possible to audit which keys are being circulated for different uids and which keys have signed them. So it would be easy to offer users a service which would tell them about new keys with one of their uids and keys that were signed with one of their keys so that erroneous signing of malicious keys can be detected and the keys revoked.

We have assumed that it is possible for users to have 'password manager' style programs, for the verification function used to verify passwords to be changed and that the GPG key-servers can be used in backwards compatible ways for new purposes. We assume this since, unlike most aspects of password-based authentication, all these things have changed before. Crucially, our proposal does not alter the input and transmission of 'passwords' through the application and therefore any change made to the server is limited to modifying the verification function.

We described a protocol that allows users to authenticate to end-points with a single message that can be generated on a device they already have without requiring input to the device. This is backwards compatible with passwords except for a change to the password verification function, which improves deployability. This provides good usability when the token can be input automatically (SOTTA) or when the user's device and the end-point both have an internet connection (OOTTA). This system could realistically be used to replace passwords in many circumstances, particularly when bootstrapping or when resolving failures. In the past, password authentication has evolved through changes in the verification function and the introduction of password managers. We hope this system will provide better authentication by evolving those parts of password authentication that have been shown to evolve in the past.

Acknowledgements Nicholas Wilson, Oliver Chick, Andrew Rice, Markus Kuhn, Robert Watson, Joseph Bonneau and Bruce Christianson all provided useful feedback on various versions of this idea.

References

[1] Anne Adams and Martina Angela Sasse. "Users are not the enemy". In: *Communications of the ACM* 42.12 (1999), pp. 40–46. DOI: `10.1145/322796.322806` (cit. on p. 2).

[2] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. "SP 800-57 Recommendation for Key Management – Part 1: General". In: *NIST special publication* (2007), pp. 1–142 (cit. on pp. 7, 8).

[3] PSLM Barreto and Michael Naehrig. "Pairing-friendly elliptic curves of prime order". In: *Lecture Notes in Computer Science: Selected areas in cryptography* 3897 (2006), pp. 319–331. DOI: `10.1007/11693383_22` (cit. on p. 8).

[4] Steven M. Bellovin and Michael Merritt. "Encrypted Key Exchange : Password-Based Protocols Secure Against Dictionary Attacks". In: *IEEE Security and Privacy*. Oakland, California: IEEE, 05/1992, pp. 72 –84. ISBN: 0818628251. DOI: `10.1109/RISP.1992.213269` (cit. on p. 13).

[5] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short signatures from the Weil pairing". In: *Journal of Cryptography* 17.4 (2004), pp. 297–319 (cit. on p. 8).

[6] Joseph Bonneau and Sören Preibusch. "The password thicket: technical and market failures in human authentication on the web". In: *The Ninth Workshop on the Economics of Information Security, WEIS*. 2010 (cit. on p. 2).

[7] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes". In: *IEEE Symposium on Security and Privacy*. 2012. DOI: `10.1109/SP.2012.44` (cit. on pp. 1, 11, 12).

[8] Mike Burrows, M. Abadi, and Roger M. Needham. "A Logic of Authentication". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 426.1871 (12/1989), pp. 233–271. ISSN: 1364-5021. DOI: `10.1098/rspa.1989.0125` (cit. on p. 6).

[9] Jeremy Clark and Paul C. van Oorschot. "SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements". In: *IEEE Symposium on Security and Privacy* (2013), pp. 511–525. DOI: `10.1109/SP.2013.41` (cit. on p. 6).

[10] Whitfield Diffie and Martin E. Hellman. "New directions in cryptography". In: *Information Theory, IEEE Transactions on* 22.6 (1976), pp. 644–654 (cit. on p. 13).

[11] Léo Ducas and Phong Q. Nguyen. "Learning a Zonotope and More: Cryptanalysis of NTRUSign Countermeasures". In: *Lecture Notes in Computer Science: Advances in Cryptology - ASIACRYPT* 7658 (2012), pp. 433–450. ISSN: 0302-9743. DOI: `10.1007/978-3-642-34961-4_27` (cit. on p. 8).

[12] "FIPS 186-3: Digital Signature Standard (DSS)". In: *National Institute of Standards and Technology (NIST)* (2009) (cit. on pp. 7, 8).

[13] Dinei Florêncio and Cormac Herley. "A large-scale study of web password habits". In: *Proceedings of the 16th International Conference on World Wide Web*. Banff, Alberta, Canada: ACM, 2007, pp. 657–666. ISBN: 9781595936547. DOI: `10.1145/1242572.1242661` (cit. on p. 1).

[14] Feng Hao and Peter Y.A. Ryan. "Password authenticated key exchange by juggling". In: *Security Protocols* 6615 (2011), pp. 159–171. DOI: `10.1007/978-3-642-22137-8_23` (cit. on pp. 10, 13).

[15] NA Howgrave-Graham and NP Smart. "Lattice attacks on digital signature schemes". In: *Designs, Codes and Cryptography* 23.3 (2001), pp. 283–290. DOI: `10.1023/A:1011214926272` (cit. on p. 8).

[16] David P. Jablon. "Strong password-only authenticated key exchange". In: *ACM SIGCOMM Computer Communication Review* 26.5 (10/1996), pp. 5–26. ISSN: 01464833. DOI: `10.1145/242896.242897` (cit. on pp. 10, 13).

[17] Neal Koblitz and Alfred Menezes. "Pairing-based cryptography at high security levels". In: *Lecture notes in Computer Science: Cryptography and Coding* 3796 (2005), pp. 13–36. DOI: `10.1007/11586821_2` (cit. on p. 8).

[18] Leslie Lamport. *Constructing digital signatures from a one-way function*. Tech. rep. October. SRI International, 1979, pp. 1–7 (cit. on p. 8).

[19] Ben Laurie, Adam Langley, and Emilia Kasper. *RFC6962: Certificate Transparency*. Tech. rep. IETF, 06/2013, pp. 1–27 (cit. on p. 5).

[20] Anil Madhavapeddy, Richard Sharp, David Scott, and Alastair Tse. "Audio networking: the forgotten wireless technology". In: *Pervasive Computing* (07/2005), pp. 55–60. DOI: 10.1109/MPRV.2005.50 (cit. on p. 9).

[21] Robert Morris and Ken Thompson. "Password security: A case history". In: *Communications of the ACM* 22.11 (1979), pp. 594–597. DOI: 10.1145/359168.359172 (cit. on p. 3).

[22] David Naccache and Jacques Stern. "Signing on a Postcard". In: *Lecture Notes in Computer Science:Financial Cryptography* 1962 (2001), pp. 121–135. DOI: 10.1007/3-540-45472-1_9 (cit. on p. 8).

[23] Kaisa Nyberg and Rainer A. Rueppel. "Message recovery for signature schemes based on the discrete logarithm problem". In: *Designs, Codes and Cryptography* 7.1-2 (01/1996), pp. 61–81. ISSN: 0925-1022. DOI: 10.1007/BF00125076 (cit. on p. 8).

[24] Colin Percival. *Stronger key derivation via sequential memory-hard functions.* 05/2009. URL: http://www.unixhowto.de/docs/87_scrypt.pdf (visited on 2014-01-07) (cit. on p. 3).

[25] LA Pintsov and SA Vanstone. "Postal revenue collection in the digital age". In: *Lecture Notes in Computer Science:Financial Cryptography* 1962 (2001), pp. 105–120. DOI: 10.1007/3-540-45472-1_8 (cit. on p. 8).

[26] Shannon Riley. *Password security: What users know and what they actually do.* 2006. URL: http://usabilitynews.org/password-security-what-users-know-and-what-they-actually-do/ (visited on 2014-01-07) (cit. on p. 3).

[27] Ron L. Rivest, Adi Shamir, and Len Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (02/1978), pp. 120–126. ISSN: 00010782. DOI: 10.1145/359340.359342 (cit. on p. 7).

[28] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell. "Stronger password authentication using browser extensions". In: *Proceedings of the 14th USENIX Security Symposium* (2005), pp. 17–31 (cit. on p. 12).

[29] C. P. Schnorr. "Efficient identification and signatures for smart cards". In: *Lecture Notes in Computer Science* 435/1990 (1990), pp. 239–252. DOI: 10.1007/0-387-34805-0_22 (cit. on p. 8).

[30] Frank Stajano. "Pico: No more passwords!" In: *Security Protocols XIX* 7114 (2011), pp. 49–81. DOI: 10.1007/978-3-642-25867-1_6 (cit. on p. 12).

[31] Daniel R. Thomas and Alastair R. Beresford. *Nigori: Secrets in the cloud.* 2013. URL: http://www.cl.cam.ac.uk/research/dtg/nigori/ (visited on 2013) (cit. on p. 12).

[32] Daniel T. Wagner, Andrew Rice, and Alastair R. Beresford. "Device Analyzer: Large-scale mobile data collection". In: *Sigmetrics, Big Data Workshop.* Pittsburgh, PA: ACM, 06/2013 (cit. on p. 2).