FIDO – that dog won't hunt

Michael Scott

Cryptographic Researcher
MIRACL Labs
mike.scott@miracl.com

Abstract. FIDO is an authentication technology based on the mathematics of public key cryptography that emerged in the 1970s and the 1980s. It is promoted by a large industry backed consortium as the two-factor successor to the username/password mechanism, which is well understood as being no longer fit for purpose. But intrinsic to FIDO is the requirement for both client-side secure hardware and a vulnerable server-side credentials database. Here we propose a better solution which would ditch both of these requirements by separating the registration and authentication processes, and which provides true multi-factor authentication using more modern ideas that have emerged from cryptographic research.

Keywords: Authentication, FIDO, M-Pin, Public Key Substitution, PKI, phishing attacks, credential databases, MIRACL

1 Introduction

On joining any organisation (server) which limits access, an individual (client) is typically issued with certain credentials, possession of which they must establish in order to gain access. Alternatively they may generate their own credentials and have them endorsed by the organisation. Either way any method of authentication which merely involves the handing over of credentials for inspection, is vulnerable to a so-called phishing attack, where they are fooled into handing their credentials to a malicious party who may then masquerade as the legitimate individual.

The problem considered here is one of a client authenticating to a server (sometimes known as the Relying Party) over the internet. We assume that a method is already in place to allow the server to authenticate to the client. There will be a one-off registation phase where credentials are established, followed over time by multiple authentication attempts.

For extra security, two-factor authentication is often recommended. It is easy to underestimate just how much harder it is for an attacker to solve two completely unrelated problems in order to achieve their malign ends, and the extent to which such a prospect demoralises all but the most persistent attacker.

2 Starting over

Consider the simplest of all authentication schemes. At registration, the client generates a memorised password (their credential), which is passed up to a server, which stores it next to the client's username in a database. To authenticate, the client sends their username and password to the server, which uses the username to look up the stored password and compares it with what has been received. It is assumed that the server is in a position to block multiple password guesses. Clearly this is not very secure, and wide open to both database hacking and phishing attacks. Basically client and server share a secret, the password. Such a protocol can come under attack at either the server or client end, or indeed during the transport of data between the two.

The first improvement is to insist on the server deploying the well known SSL/TLS protocol to ensure that an encrypted tunnel is created between client and server, which effectively solves the transport issue and authenticates the server to the client. Briefly the server has a private/public key pair, and a client generates an ephemeral session key, encrypts it with the public key, and sends it up to the server who can recover it. To convince the client that they are using the right public key, they do not just take it on trust from the server, but rather extract it from an X.509 certificate which has been digitally signed by a recognised Certificate Authority (CA). This is well known tried-and-tested Public Key Infrastructure (PKI) solution which ensures server integrity and allows e-commerce to flourish on the Internet.

Unfortunately this is not enough to prevent sophisticated phishing attacks, as the wrong server can authenticate itself just as easily as the right one, if the client is not careful to establish exactly who they are talking to. We will refer to this method as the proto-authentication protocol.

3 Introducing client-side secure hardware

Now let us allow ourselves a secure hardware vault on the client side. This allows the client to store inside the vault a very large random 128-bit secret, that he does not need to memorise, or indeed ever be aware of. At registration it is transported to the server and stored next to their username. Now we can be a little cleverer. To authenticate, the server sends a random challenge to the client which gets its vault to encrypt it using the secret as a key, sends it back to the server which looks up the same secret from the database and uses it to decrypt. If the result is the same as the original challenge, authentication succeeds.

Note that we have come a long way thanks to secure hardware. The client experience is now passwordless, phishing attacks are impossible (as the challenge is different every time), and the client secret is no longer transmitted to the server with every authentication, in fact it never leaves the security of its vault. We have established possession of the secret without revealing anything useful about it. If a two factor authentication experience is desired – easy – arrange it such that the vault only performs its function on entry of a short PIN number. We

note in passing that once a secure vault is assumed, it may also be used to store a biometric template of the client, so that presentation of that biometric would be an alternate mechanism for opening the vault. For the purposes of this paper we will assume the PIN entry option is used as a second factor.

Since client and server are intent on entering into a trusting relationship, it doesn't really matter which of them generates the large secret. So a simple elaboration would be for the client hardware to generate an ephemeral public/private key pair, and during registration send the public key to the server, which generates the secret, encrypts it with the public key, and sends it back to the hardware, which decrypts it with the private key that can then be immediately deleted. Now the secret is never visible at any endpoint on the client side.

Note that this simple idea gives us everything that FIDO gives us on the client side, at a fraction of the cost and complexity.

However it is not so clear that all is well on the server side. That credential database is still a major weakness. The mutual client/server secret is just sitting there, in the clear and inviting attack by a resourceful hacker.

3.1 Username/Password

The Username/Password protocol is another way of improving on our proto scheme, this time without requiring secure hardware. Now the basic idea is that a one-way hash of the password is stored on the server side, and when the client password is presented to the server, it is the hash of it that is compared with the value stored in the database. This certainly improves things at the server end, as passwords are no longer stored in their original form, and cannot easily be recovered from the hash because of its one-wayness.

In fact there is often a lot more than that to Username/Password as deployed by organisations like Facebook [11] and LinkedIn [8]. Commonly the stored credential is a combination of a salt and an iterated hash of the concatenation of username, salt, pepper, a secret key and the actual password. Such a method is recommended by NIST [2] section 5.1.1. Each component adds something to the overall security. By iterating the hash function maybe 10,000 times we introduce a tiny but acceptable delay into the authentication process, but we also increase the attackers workload by the same factor. The salt, typically 128 random bits, ensures that identical passwords do not result in identical credentials, and again slows down a brute-force attack which iterates through a large dictionary of common passwords, forcing the attacker to search the full dictionary as it attacks each individual account. Care is also taken to protect against active substitution attacks where an attacker can overwrite his victim's credential with one of his own choosing. By including the username in the hash, the attacker cannot simply substitute his own genuine credential for that of his victim.

Perhaps the most significant component is the "pepper". This is a global secret, typically stored in a HSM (Hardware Security Module), which must be contributed on-the-fly before the correct hash is calculated. It is there to mitigate

against the most common failure mode of Username/Password, where the credential database is stolen, and an attacker can at their leisure go about attacking every individual's password. Without possession of the pepper component, the attacker cannot succeed.

Safely handling a single global secret that is so widely used is problematical but it is an option for larger, well equipped corporations. Even if the pepper is compromised (perhaps by an insider attack), the salting provides a second level of security. As a last line of defence, the choice of a strong password can still thwart the determined attacker.

We conclude that Username/Password, certainly if implemented carefully with the "pepper" component, is still capable of providing a secure authentication experience. Alas it is intrinsically one-factor, and strong passwords are, to put it mildly, inconvenient.

4 FIDO

FIDO was first proposed in 2013 by an alliance of industry leaders concerned by the shortcomings of Username/Password, and motivated by an acceptance that two factor authentication provides much more security.

Much progress has been reported on the standardisation and specification fronts. Adoption of the technology by large corporations and alliance members as an optional authentication mechanism has accelerated. However, from anecdotal evidence it would appear that adoption by edge users has been slower, and the hoped-for phasing out of username and password largely hasn't happened to date. One promising development has been the recent release of an open source implementation by Google to allow the creation of FIDO enabled hardware authentication tokens [3].

FIDO attempts to improve upon our original hardware based solution by using public key cryptography. The client vault generates a long term public/private key pair, and on registration supplies its username and public key to the server, which stores them in a credentials database. To authenticate, the client presents its username and the server responds with a random challenge. The client uses its private key to digitally sign this challenge, and sends this signature back to the server, who verifies it using the public key it extracts from the credential database. We acknowledge that there is a lot more to FIDO than this simple description. In particular there is an attestation mechanism used to attest to the properties of the secure hardware and its capabilities. However this is separate from the authentication process considered here. There is a lot of engineering involved in implementing FIDO. Browsers are modified to allow communication to/from a hardware authenticator via quite complex WebAuthn APIs, CTAP, and CBOR protocols.

But already we can see the big advantage: The credential database stores public keys rather than secrets or hashed passwords, which on the face of it sounds safer in the context of the hacker who steals it. Determining private keys from public keys is a much more daunting task. As before the client has

authenticated without handing over its secret (the private key), and since the challenge is different every time, a phishing attack is again no longer possible.

The vault may exist as an authentication token that connects via USB to a laptop or desktop computer. More likely it exists as an extension to the processor architecture used by a modern smart mobile phone.

Since FIDO uses public key cryptography it might be assumed that therefore it also uses a PKI, as such is a normal requirement to ensure correct handling of public keys. The main problem solved by PKI is that of making the association between the public key and its true owner. In the absence of such a mechanism we have no way of knowing that the public key we are using is the right one, and therefore we are wide open to a public key substitution attack. However while FIDO could be integrated into an existing PKI scheme if one already exists for clients, this is not the recommended approach [7].

4.1 The problem with PKI

PKI has a well deserved reputation for being difficult for clients to manage. Consider the email experience: most email clients offer a facility for clients to use PKI to encrypt and digitally sign their emails but as we know, hardly anyone uses this facility (unless mandated within large enterprises with large IT support departments). To the surprise of many, PKI encrypted/signed email never achieved mass adoption. For organisations which do support PKI down to the full enrolment of each individual client, the authentication problem is already largely solved, and there would be no need for FIDO – "Why use FIDO at all?" [7].

To avoid suffering the same fate adoption-wise as PKI encrypted emails, it is understandable that FIDO avoids the use of a full PKI [7]. So we have a problem here: A full PKI would have a negative impact on adoption, while anything less will introduce security weaknesses. Larger enterprises appear to be aware of the danger, and have gone to some lengths to integrate FIDO with existing PKI infrastructure [13]. But this will not be the norm as is made clear in [7], and certainly will not apply to the consumer market. There appears to some confusion around this issue. Some proponents of FIDO seem to be under the mistaken impression that FIDO always deploys a full PKI [16]. It doesn't (although confusingly PKI is used for the seperate attestation process). As stated quite baldly in [7] "FIDO does not use digital certificates for users". Or as coyly stated in the website FAQ "FIDO takes a lightweight approach to asymmetric public-key cryptography".

Summarising [7] says "Broadly, FIDO is simpler (than PKI) for end-users to use, and for application developers to integrate into web and mobile applications. FIDO infrastructures can also be less expensive to operate and manage given that they do not require many artefacts generally associated with PKIs". See page 8 of this FIDO document to see which artefacts are left out.

4.2 So how does FIDO protect public keys? Spoiler – it doesn't.

Recall that a FIDO server is expected to maintain a database of user credentials, basically a credential ID and the associated public key that matches the private key stored back in the client's vault. Actually FIDO public keys are not really "public" in the usual sense. They reside in the server's database, which would not itself normally be visible to the curious onlooker. Read access to the credential database is regarded as a "threat" in the most recent FIDO security reference draft [6].

The WebAuthn guide states that credential databases "are no longer as attractive to hackers, because the public keys aren't useful to them.", which is hardly an encouragement for the implementer to go to any great lengths to protect them. This leads to a classic false sense of security. "Databases are no longer targets because public keys are useless without the corresponding private keys" [17]. Not so fast!

The most devastating attack on the FIDO credential database is a public key substitution attack. The attacker inserts their own public key for that of their victim, and hence gains access to their account, which is just as devastating as a password file compromise in the username/password context. The credential database, no matter what its format, is always a tempting target for attack, as success compromises not just individual accounts, but all accounts. Such substitution attacks are already taken into account by the Username/Password community – see above. It is surely ironic that while they do take the issue of credential substitution seriously, the FIDO community apparently does not.

As explicitly admitted by FIDO's own Security Reference document [6] section 7.3.1/2.1.2, an attacker who gains write access to the server database can launch a public key substitution attack. The outcome results in a violation of FIDO's Strong User Authentication, which is of course the whole point of FIDO. The FIDO response is revealing. The attack is described as "outside of the scope of the FIDO specifications" [6]. So it is not an issue for FIDO, and if such an attack is successful it would not be regarded as a failure of FIDO. This does not, of course, make such an attack any less of a problem. Someone has to make sure this doesn't happen, but noone has any idea how to go about it other than to implement a fully fledged PKI, which FIDO itself deprecates. We note in passing that FIDO is explicitly ignoring the NIST recommendation [2] section 5.1.8.2 which states that public keys "SHALL be protected against modification".

In fact there is a growing awareness of the problem, and [7] can be read as a somewhat pained attempt to square this circle. "The implementation of FIDO can benefit organizations in many ways. It offers strong public-key based authentication that is equivalent to certificate-based authentication but without the overhead of maintaining complex and expensive public-key infrastructure." One of the authors of [7] himself has recognised and highlighted the threat [12], there described as the "substitution of keys" attack.

It is worth quoting his contribution to this discussion group thread [4] at length "This is why I shudder to think of the consequences for FIDO when I see companies who are not in the cryptographic key-management space (banks,

e-commerce companies, CRM companies, etc.) who are under the mistaken impression that if they simply implement a protocol and put together a GUI on the implementation, they have themselves a FIDO Server. Who are they going to blame when their FIDO-based strong-authentication application is compromised? Themselves or FIDO?"

In his own supported product [15] as is made clear from this discussion thread, an attempt has been made to mitigate by deploying a cut-down PKI-like solution, where every credential is digitally signed by the server management (analogous to the "pepper" idea in Username/Password). But the difficulties and shortcomings of this idea are also emphasised. And a quick search through some other open source FIDO server implementations makes it clear that the problem is not widely recognised, and countermeasures are non-existent.

Our experience with standards is that unless certain actions are mandated, they will not be implemented. The reality is that many existing FIDO credential databases are wide open to this attack.

4.3 Is FIDO truly two-factor?

As stated above the client's experience is undoubtedly two-factor. But on closer inspection we can see that in reality it is really two-step. The PIN (first factor) must be entered first to release the private key (second factor). Since the second factor is stored in hardware, the hardware itself may also be considered as the second factor (possession of the authentication token, or of the mobile phone).

Clearly an attacker who can penetrate the vault, does not need the PIN. Therefore in reality the process is only protected by a single factor, that being the integrity of the hardware vault.

4.4 Is secure hardware secure?

Secure hardware certainly seems like a good idea, and potentially provides an important resource for cryptographers. It provides a safe place to store keys and other sensitive material (like biometric templates). It also promises to implement the cryptography faster. But there are problems.

- Hardware cannot be easily patched.
- Hardware cannot be easily replaced.
- Can we trust in the security of the hardware?
- Can we trust that any back-doors will not be exploited?
- Hardware is expensive.

Rather than using secure hardware to protect a secret, an alternative idea is to simply split up the secret, an approach very much in tune with the whole idea of multi-factor authentication.

5 FIDO – our verdict

Almost all of the benefits of using FIDO arise from the assumption of the existence of secure hardware on the client side. The use of public key cryptography technology from the 1980s is unimaginative, and clearly not fully understood by many of FIDO's proponents. The internal contradiction highlighted above has never been properly addressed. The advantage over a simple shared secret scheme (with secure hardware and peppered credentials) is in fact minimal.

Most shocking of all is that the most common security threat to username/-password, an attack on the credential database and the damage that can be done as a result, has not been addressed by FIDO, which washes its hands of responsibility by declaring it as being "outside of the scope" of its efforts.

6 An alternative approach

Next we briefly suggest an alternative solution called M-Pin [9], based on bilinear maps on elliptic curves, which is a technology which has proven to be a vehicle for many novel cryptographic constructs [10]. First we identify the main problems as being the credential database and the requirement for secure hardware. Let us dispense with both and get ourselves a simple software-only 2-factor replacement for username/password.

6.1 Client side, look – no hardware!

Consider now that a user's identity (username) is hashed and mapped to a point P of large prime order on an elliptic curve, and that the user is issued with a secret sP, by some trusted authority (TA) who possesses a master secret s. The user then chooses a PIN number α and splits their secret into two factors $(s-\alpha)P$ and α. It is then a trivial matter to recombine these factors to restore sP. It should also be clear that an attacker who gains access to one of these factors cannot determine the other. What is not so immediately obvious is that an attacker who also gains access to the extra information xP and xsP for some unknown x, also cannot determine one factor from the other, as each case reduces to finding a solution to the Decisional Diffie-Hellman (DDH) problem, which is known to be hard in prime order elliptic curve groups of a suitable size.

The form of these secrets allows them to split easily into multiple factors, and hence they offer an ideal vehicle for multi-factor authentication. Furthermore the TA that issues these secrets can also split its responsibilities by trivially observing for example that $sP = s_1P + s_2P$, where s_1 and s_2 are generated and maintained by two independent entities under separate control, known as DTAs (Distributed Trusted Authorities). Now the full master secret s does not exist anywhere in the system, and both DTAs would need to be hacked in order to access it.

Next let us consider a signature scheme where the signature of a challenge c is created by generating a random x and consists of the tuple {Username,U =

xP,$V = (x + c)sP\}$, which can be verified by the server. Before we describe how such a signature is verified, let us recap on the client's experience. First the client registers with the DTAs and is supplied with its secret, which it immediately splits into a stored blob of data $(s - \alpha)P$ and a memorised password α. We emphasis that the blob of data does not need to be stored in secure hardware as it derives its security from the fact that it is incomplete and useless due to the missing PIN component. This is true two factor authentication, not two step. When required to authenticate by signing a challenge, the PIN part is restored and the signature created. As before we assume that the server is pro-active in blocking PIN guessing attacks (which present as failed authentication attempts).

6.2 Server side, look – no credential database!

Now we assume that the elliptic curve chosen is actually a type-3 pairing-friendly curve [10]. Recall that a bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are elliptic curve groups, and $\mathbb{G}_T$ is a finite extension field [10]. The client side action takes place entirely in $\mathbb{G}_1$ and is unaffected by the pairing-friendly nature of the curve. The server is issued by the DTAs with a single secret sQ, where Q is a fixed point in $\mathbb{G}_2$. This secret point can be considered as a global public key that works for all clients, and replaces the entire FIDO database. Verification of the signature described above is carried out by independently hashing and mapping the username to the point P, and testing if

$$e(V, Q) \stackrel{?}{=} e(U + cP, sQ)$$

This signature scheme is a provably secure method first described by [1] and [5]. The PIN extraction idea comes from [14]. That the DDH assumption applies in the group $\mathbb{G}_1$ is the well established XDH assumption.

7 Conclusion

FIDO was a noble attempt to come up with a 2-factor replacement for the legacy username/password method of authentication. Unfortunately its internal contradictions were never properly recognised, and since they are intrinisc to the primitive public key cryptography that underpins it, we would predict that its adoption will eventually stall.

Secure hardware, while potentially a great asset, has its own issues. Amongst these there is a major trust issue, highlighted by the current Huawei controversy. Cryptographers are coming to the conclusion that splitting a secret and/or adopting methods from Multi-Party Computation[1] may be a better way to protect a secret.

Credential databases are notoriously difficult to defend, and are best done away with if at all possible. Here we point out that this is indeed possible, and suggest one concrete solution.

[1] https://www.unboundtech.com/

References

1. M. Bellare, C. Namprempre, and G. Neven. Security proofs for identity-based identification and signature schemes. In *Eurocrypt 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 268–286. Springer-Verlag, 2004.
2. P. Grassi et al. NIST special publication 800-63b digital identity guidelines, 2017. https://pages.nist.gov/800-63-3/sp800-63b.html.
3. Google. OpenSK is an open-source implementation for security keys written in Rust that supports both FIDO U2F and FIDO2 standards. https://github.com/google/OpenSK.
4. Google Groups. Why are the already registered keys needed during registration?, 2016. https://groups.google.com/a/fidoalliance.org/forum/#!forum/fido-dev.
5. K. Kurosawa and S-H. Heng. From digital signature to ID-based identification/signature. In *PKC 2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 125–143. Springer-Verlag, 2004.
6. R. Lindemann. FIDO security reference, 2018. https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-security-ref-v2.0-id-20180227.pdf.
7. S. Machani and A. Noor. FIDO enterprise adoption best practices, 2019. https://fidoalliance.org/white-paper-fido-and-pki-integration-in-the-enterprise/.
8. A. Mani. Life of a password. Real World Cryptography – 2015. https://www.slideshare.net/ArvindMani1/amanirwcpassword.
9. Miracl. MIRACL Trust Multi-Factor Authentication, 2020. https://miracl.com/miracl-trust-multi-factor-authentication/.
10. N. El Mrabet and M. Joye, editors. *Guide to Pairing-Based Cryptography*. Chapman and Hall/CRC, 2016. https://www.crcpress.com/Guide-to-Pairing-Based-Cryptography/El-Mrabet-Joye/p/book/9781498729505.
11. A. Muffet. Facebook: Password hashing and authentication. Real World Cryptography – 2015. http://bristolcrypto.blogspot.com/2015/01/password-hashing-according-to-facebook.html.
12. A. Noor. FIDO-enabling a web-application using Universal 2nd Factor (U2F), 2015. https://fidoalliance.org/wp-content/uploads/FIDO-enabling-web-applications_Noor.pdf.
13. M. Queralt. Extending public key infrastructure for mobile users via FIDO universal authentication framework, 2019. https://medium.com/@caumike/password-less-authentication-extending-public-key-infrastructure-for-mobile-users-via-fido-3a0da263b385.
14. M. Scott. Authenticated ID-based key exchange and remote log-in with simple token and PIN number. Cryptology ePrint Archive, Report 2002/164, 2002. http://eprint.iacr.org/2002/164.
15. StrongKey. Open-source FIDO server, featuring the FIDO2 standard. https://github.com/StrongKey/fido2.
16. S. Tzur-David. Your complete guide to FIDO, FIDO2 and webAuthn. https://doubleoctopus.com/blog/your-complete-guide-to-fido-fast-identity-online/.
17. Yubico. Webauthn introduction. https://developers.yubico.com/WebAuthn/.