M-Pin: A Multi-Factor Zero Knowledge Authentication Protocol

Michael Scott

Chief Cryptographer
Certivox Labs
mike.scott@certivox.com

Abstract. Here we introduce the M-Pin client-server protocol, which features two-factor client authentication as an alternative to Username/Password. Despite the mathematical complexity of the protocol we demonstrate that an M-Pin client can be implemented in an environment with limited computational capability.

1 Executive Summary

The M-Pin protocol is intended to replace the well-known Username/Password authentication mechanism which is widely considered to be effectively broken. The main problem is the existance of a "password file" on the server, which is commonly stolen and hacked, revealing most user passwords.

The idea behind M-Pin is that each registered client is issued with a large cryptographic secret. They then prove to a server that they are in possession of this secret using a zero-knowledge proof. This removes the requirement for any information related to client secrets to be stored on the server.

Industry commentators have long advocated a multi-factor solution. The novel feature of M-Pin is that the cryptographic secret may be safely split up into any number of independent factors. Each of these factors has the same form, they are points on an elliptic curve. To recreate the original secret they are simply added together again – its as simple as that. One factor might be derived from a short 4-digit PIN. Another might be a "token" conveniently dropped into the users browser.

Classic two-factor solutions are in fact often hierarchical and two-level. A key generated from one factor is used to unlock the other. Disjoint technologies are used by each factor. Typically a password (or a biometric) might be used to unlock an authentication key stored in a file. Strictly speaking this solution is only one-factor, as it is only this authentication key that is required, and an attacker would be satisfied if they could obtain this without knowing the password. However since this is probably not possible we accept that the overall effect is two-factor authentication. Software encryption might be used as the locking mechanism, but since a brute force attack will discover the authentication key, the password must become a large hard-to-remember pass-phrase. The alternative (which achieves the same functionality as two-factor M-Pin) is to lock the authentication key into a secure hardware vault. Now a short PIN can be used to unlock it. However secure hardware is expensive and may not be supported on all devices. Another downside of this classic approach is that the extension to multi-factor authentication is not at all obvious.

2 Introduction

A strong client-server protocol should (a) authenticate the client to the server, (b) authenticate the server to the client, and (c) should result in a negotiated encryption key with which subsequent communications can be encrypted.

The standard method of implementation uses a Username/Password mechanism to authenticate the client to the server, and the well known TLS/SSL protocol to authenticate the server to the client and to establish the encryption key. The weakest link here is the Username/Password mechanism which is widely regarded as being broken. SSL itself, to a lesser extent, has been weakened by intensive scrutiny which has revealed some exploitable vulnerabilities.

To replace Username/Password, multi-factor authentication is the most often touted solution. Of all the possible form-factors the simple ATM-like combination of a token and a PIN number is the most user-familiar and user-friendly. But until now (surprisingly) no sophisticated cryptographic protocol was available to support it, other than those that used simplistic methods derived from classic symmetric cryptography. Implementations invariably involved some kind of (potentially expensive) hardware token. Cryptographic solutions based on modern asymmetric cryptography suffered from weaknesses which allowed an attacker, in possession of some easily accessible side information, and who had captured a software token, to mathematically immediately determine the associated PIN number. Therefore they were not genuinely two-factor.

Here we describe the M-Pin technology solution which simply replaces Username/Password while continuing to leverage SSL.

An important aspect of M-Pin is the involvement of a third party, called the Trusted Authority. In classic Username/Password schemes the client registers directly with the server, who maintains an "encrypted" (in fact hashed) file of client passwords. So the server is not just responsible for day-to-day operations, it is responsible for client registration as well. Note that this is in contrast with (the much more successful) SSL protocol, which already involves a third party in the form of a certificate issuing authority (CA), such as Verisign. The CA registers the server by issuing on its behalf a public certificate. This certificate based technology is known as the Public Key Infrastructure (PKI).

So with the likes of SSL, server registration is quite separate from day-to-day server operation. With M-Pin we do something similar for the client – we separate out the day-to-day client-server functionality from client registration, which is now handled by a Trusted Authority (TA). So a TA is to M-Pin what a CA is to SSL/PKI. One of the big benefits of this approach is that a break-in to the server does not cause nearly as much damage, as long as the TA remains inviolate. Just as hacking a single SSL server is not nearly as bad as hacking

Verisign. In particular, using M-Pin, no client secrets, or values derived from client secrets, are stored on the server.

3 Pairing-Based Cryptography

To realise our solution we will be exploiting the relatively new (but rapidly maturing) science of pairing-based cryptography (PBC). Pairing-Based Crypto provides an extra structure which often allows solutions to complex problems that proved intractable to the standard mathematics of Public-Key Cryptography. The poster child for PBC was Identity-Based Encryption, whereby the identity of a client became their public key. The idea was around for a long time, but traditional cryptographic primitives failed to produce a solution. However with the introduction of PBC solutions were found almost immediately [5].

A Type-3 pairing [10] is a mapping $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are all of the same prime order q. A pairing works on a special pairing-friendly elliptic curve [3], [9]. For a BN curve $\mathbb{G}_1$ are points on the curve over the base field $\mathbb{F}_p$, $\mathbb{G}_2$ are points on the sextic twist of the curve over the quadratic extension field $\mathbb{F}_{p^2}$, and $\mathbb{G}_T$ are elements in the cyclotomic subgroup embedded in the finite extension field $\mathbb{F}_{p^{12}}$. The pairing itself is written as a function with two inputs $C = e(P, Q)$, where $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ and $C \in \mathbb{G}_T$. A BN pairing-friendly curve is a perfect fit for security at the AES-128 level, and so we will assume its use here. The most important property of the pairing is its bilinearity:

$$e(aP, Q) = e(P, Q)^a = e(P, aQ)$$

To create a client-server protocol, it is important that client and server secrets should be kept distinct. A simple way to exploit the structure of a Type-3 pairing is to put client secrets in $\mathbb{G}_1$ and the server secret in $\mathbb{G}_2$. For a Type-3 pairing there is assumed to be no computable isomorphism between these groups, even though both are of the same order.

4 Of Keys and Cucumbers

One of the novel aspects of pairing-based cryptography is that deployed secrets are commonly represented as points on an elliptic curve, which are the result of multiplying a known point by a master secret s. So for example a secret might be of the form sP, where P is known. There are a number of interesting things we can do with secrets that have this form, that are not possible with the secrets that arise when using other cryptographic technologies. For example they can be split into two, into s_1P and s_2P where $s = s_1 + s_2$ and $sP = s_1P + s_2P$. In fact they can be just as easily split into multiple parts, just like chopping up a cucumber. We can also add extra components to create a secret of the form $s(P_1 + P_2) = sP_1 + sP_2$. It is the flexibility that arises from this form of the secret that allows us to introduce the idea of chopping off a tiny sliver of the secret to support a PIN number. It also facilitates the concept of "time permits" as described below.

5 The Trusted Authority

The Trusted Authority will be in possession of a master secret s, a random element of $\mathbb{F}_q$. A client secret is of the form $s.H(ID)$, where ID is the client identity and $H(.)$ a hash function which maps to a point on $\mathbb{G}_1$. Here we follow [18] and assume that H is modelled as a random oracle where $H(ID) = r_{ID}.P$ where $r_{ID} \in \mathbb{F}_q$ is random and P is a fixed generator of $\mathbb{G}_1$. The server will be issued with sQ, where Q is a fixed generator of $\mathbb{G}_2$. Note that this will be the only multiple of s in $\mathbb{G}_2$ ever provided by the TA. Servers will always be associated with their own unique master secrets.

Note that the TA functionality can be trivially distributed using a secret sharing scheme, to remove from the overall system a single point of failure. In the simplest possible case there may be two Distributed Trusted Authorities (DTA), each of which independently maintains their own share of the master key. So $s = s_1 + s_2$, and each DTA issues a part-client secret $s_1 H(ID)$ and $s_2 H(ID)$, which the client adds together to form their full secret. Now even if one DTA is compromised, the client secret is still safe.

6 Splitting the client Secret

An important idea is that a PIN number α can be securely extracted by the client from their secret, to create the token [16]. Our two factors of authentication are then this memorised PIN and the token. To be explicit, for an individual Alice whose identity hashes to a point $A \in \mathbb{G}_1$, her full secret as issued by the TA is the point sA. This is split into the token $(s-\alpha)A$, and the chosen PIN number α. The token is created by calculating αA and subtracting it from sA. The full secret can be reconstructed from its two components by Alice as $sA = (s - \alpha)A + \alpha A$.

7 M-Pin

This protocol builds on a long history of Identity-Based identification (IBI) protocols, that originate with the seminal work of Fiat and Shamir [8]. The basic idea is for a Prover to identify itself using an identity-related secret issued by a Trusted Authority, while revealing nothing of that secret to the Verifier. This is often called a zero-knowledge proof, as proof of possession of the secret is established while revealing nothing of the secret itself. It is important to emphasise that an identification protocol results only in the Verifier (in this case the server) either accepting or rejecting the Prover (in this case the client).

IBI protocols were studied in depth by Bellare et al. [4] as part of a larger framework whereby many standard identification (SI) methods could be "surfaced" as IBI schemes, or indeed as full Identity Based Signature (IBS) schemes. By "pushing" some proposed (but unproven) IBS schemes down to their underlying SI description, there were able to resurface them, but this time with full security proofs. They considered some pairing-based schemes, the best of which appears to be that which first appeared as an IBS scheme proposed independently in [7] and [20]. Independently (of [4]), Kurosawa and Heng [13] came up with more-or-less the same idea. All of this research owes a debt to the original (non-identity based) short signature scheme of Boneh, Lynn and Shacham [6].

Note that in standard IBI protocols the Verifier has no secrets. The Prover is simply trying to prove that they are entitled to their claimed identity, to anyone that is interested. Here however only the unique Verifier in possession of the secret sQ is in a position to carry out the verification. We need this restriction, as otherwise a corrupt Verifier could be used as an Oracle against which to test PIN guesses by someone who had stolen the associated token. Therefore we assume that the server which is running SSL or its equivalent, also takes responsibility for protecting sQ. The only other change we make is to split client secrets into token and PIN as described above.

To place the server secret sQ in context, observe that in standard PKI everyone has their own identity and an independent public/private key pair. So there is a public key for every individual, bound to their identity via a certificate issued by a CA. On the other hand in ID-based cryptography everyone has an identity, and a private key derived from that identity by a TA. And there is just one public key, the TA's public key. This TA public key is used in various protocol-dependent contexts. In the M-Pin protocol the TA public key becomes the server secret. This means that only the server can authenticate one of its own clients. But this is a perfectly natural requirement in the authentication context.

The basic M-Pin protocol is shown in Table 1. Observe that on the client side all of the computation is in the simpler group $\mathbb{G}_1$. On the server side the product of two pairings can be calculated using standard multi-pairing methods [17], and will cost much less than two separate pairing computations. The correctness of the protocol can be quickly established using the bilinearity property.

7.1 Protocol Outputs

At the end of a protocol run the client still does not know whether or not they have succeeded. This is important as we now need to take action to handle unsuccessful connections appropriately. In a real implementation this responsibility might be passed off to another non-cryptographic process, which is informed by the available protocol outputs.

What outputs should a protocol like M-Pin return after completion? For Username/Password the outcome is basically to either allow access or refuse it, but also some simple mechanism in place to block on-line password guessing attacks. Our response can at a minimum be similar to this, but possibly more nuanced.

In the event of a successful M-Pin outcome, there is not much more to be done. The client must have been registered correctly with the Trusted Authority, and therefore has been issued with a valid token and has input a valid PIN, associated with their claimed identity. In the event of protocol failure at first glance there appears to be relatively little for the server to work with to formulate

Alice - identity ID_a	**Server**
Generates random $x < q$	Generates random $y < q$
$A = H(ID_a)$	
$U = xA$	
$ID_a, U \quad \rightarrow$	
	$\leftarrow y$
$V = -(x + y)((s - \alpha)A + \alpha A) \rightarrow$	
	$A = H(ID_a)$
	$g = e(V, Q).e(U + yA, sQ)$
	if $g \neq 1$, reject the connection

Table 1. M-Pin

an appropriate response. But for a two-factor authentication scheme like M-Pin there is at a minimum a vital requirement for the server to implement a 3-strikes-and-you-are-out strategy to prevent an attacker who has captured the token from trying all possible PINs until they hit upon the right one. The server does not necessarily even have a list of registered users – it would be natural that such a list should be maintained and managed by the Trusted Authority.

It may help the server to decide on an appropriate response if (a) they knew if the failing client had a valid token, and (b) if they knew the extent of the error in the entered PIN. In fact there is a simple way to derive from a failed protocol run the extent of the PIN error – and if this is outside of the range of valid PINs, then this implies that the "client" does not in fact have a valid token.

Observe that the M-Pin protocol depends on both the client and the server having the same value of the master secret s involved in their calculations. If a client enters an incorrect PIN, it is as if they have instead used $s + \delta$ in their calculation, where δ is the extent of their error. The server can then search for δ by iterating through all the possiblities for $sQ+\delta.Q$ on its side, until the protocol outcome is correct. By exploiting the bilinearity of the pairing, this can be done in an obvious way requiring only one multiplication in $\mathbb{G}_T$ for each candidate δ.

However we can do even better. If the final step of the protocol results in a value of $g \neq 1$, then in fact $g = e(U+yA, Q)^{\delta}$. To find δ requires the calculation of a pairing and the solution of a discrete logarithm problem in $\mathbb{G}_T$. The appropriate algorithm is the method of Pollard's Kangaroos [14]. This is a "square root" algorithm, which means that for a 4-digit PIN only a few hundred multiplications in $\mathbb{G}_T$ will be required to find δ, which is completely practical.

In the case where the extent of the PIN error can be determined by the server (and this potentially applies to any two-factor authentication scheme, but in particular to the protocols suggested here), a more intelligent response is possible rather than the classic and simplistic "3-strikes-and-you-are-out" mechanism.

A server might attempt to intelligently distinguish between a bad entity who has captured the token and is trying to find the PIN by trying all possible combinations, and the good entity who has either mistyped their PIN, or inadvertently typed in the wrong PIN. One simple way to exploit knowledge of PIN error is to not punish again a user who enters the same wrong PIN more than once. Note that it is of no value to a bad entity to guess the same PIN twice, so nothing is lost by doing this.

In the case of a mistyped PIN, the error will typically be in only one digit. Again this can be detected by the server from the PIN error. Therefore we suggest an example of a more elaborate scoring mechanism. Here a user would only be locked out if they reach a score of more than 10. A completely wrong PIN scores 4, the same wrong PIN entered again scores 0, a PIN out by just 1 digit scores 2, and a PIN out by just 2 digits scores 3. In this way a genuine good entity will struggle to reach a score of 10, whereas a bad entity will typically get there after just 3 attempts.

Note that other side-information will be available to the server which can be folded into an intelligent decision on whether or not to lock out a particular identity. The server will also know the IP address of the client, the browser they are using, and the time of the attempted log-in. Combined with the identity and PIN error this amounts to the who, what, where and when of each failed connection.

7.2 Cryptographic Security

Here we will briefly consider the cryptographic security of the proposed scheme. The underlying IBI scheme we are using was proven secure by [4], in that breaking the scheme is shown to be as difficult as the one-more Computational Diffie-Hellman problem. They considered the protocol in the context of a Type-1 pairing (for which $\mathbb{G}_1 = \mathbb{G}_2$). This can also be considered as a Type-3 pairing in which there exists a computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. The problem of transferring a security proof from a Type-1 to a Type-3 pairing was considered by Smart and Vercauteren [18]. Their simplest solution was to "relativise" the security proof in the context of an Oracle which was made available to an attacker and which implemented the isomorphism. The fact that such an isomorphism is not known, evidently does not weaken the proof.

Our next concern is that an attacker who captures a token should not be able to determine the associated PIN number without the willing participation of the server. A particularly powerful attacker would be one who was himself a client of the server, or who was able to recruit a coalition of insiders willing to provide their full client secrets.

So consider an attacker who was in possession of sA, sB, sC ... and the victims token $(s-\alpha)Z$. Would it be possible to continually add Z to the victims token and to distinguish the case when it too became equal to sZ? This is exactly the Generalized Decisional Diffie-Hellman problem as considered by Bao, Deng and Zhu [2]. They proved that this reduced to the standard DDH (Decisional Diffie-Hellman) assumption in the group $\mathbb{G}_1$.

That the DDH problem is hard in $\mathbb{G}_1$ on a Type-3 pairing is known as the XDH assumption, first informally implied in [16], and made explicit in [1].

Finally consider an attacker who has access to the victims token and manages to eavesdrop a run of the protocol, or perhaps succeeds via a "phishing" attack to get the client to engage in a protocol run directly with it. Such an attacker can harvest the values zA, $(s-\alpha)A$, and zsA, the first and last values obtained from the protocol run. However to find the PIN requires the attacker to be able to distinguish zA, sA and zsA from three values without this relationship, which is again covered by the XDH assumption.

Note that the full client secret is reconstructed from the token and the PIN before it is used in the IBI protocol. Also the server does not transmit anything to the client which might be of use in determining a PIN – in fact it only transmits the random challenge y.

7.3 Securing the Server

Consider now a successful break-in by a hacker into the server. Before considering the implications of this, it is worth pointing out that the server secret, (independent of the number of clients is supports), is just the single point $sQ \in \mathbb{G}_2$. Therefore it would be easy to protect this secret inside of a Hardware Security Module (HSM). However assuming that this secret is captured, this would allow the hacker to set up a false server to whom clients could be attracted. It would also allow the hacker to find the PIN associated with a captured token. In fact this must be true of any two-factor authentication scheme, as knowing server secrets, a false server can be created and PIN guesses can be tested against it. However the successful hacker is not able to reconstruct full client secrets, and so cannot itself log onto the genuine server and into client accounts. So the potential for mischief is greatly reduced.

We can (if we want) do better. Recall the server secret can be issued by the DTA in two parts, s_1Q and s_2Q which are added to create the full server secret sQ, where $s = s_1 + s_2$. In fact the protocol can be completed on the server side while keeping s_1Q and s_2Q separate.

On the server side the part of the calculation involving the server secret is of the pairing $e(X, sQ)$. for some random X. But by the magic of bilinearity $e(X, sQ) = e(X, s_1Q).e(X, s_2Q)$.

So for example the server could have two HSMs (each from a different manufacturer). One would store s_1Q and the other s_2Q. One would calculate $e(X, s_1Q)$ and the other $e(X, s_2Q)$. The server process would then simply multiply these values and continue with the protocol as normal. However knowing $e(X, s_1Q)$ for example, and knowing X, does not reveal anything about s_1Q. That is the reverse pairing problem which is believed to be hard. So no single entity ever knows the server secret sQ, and a possible single-point-of-failure is eliminated.

7.4 Discussion

The described scheme could as suggested be run under the protection of SSL on the server side. However SSL is not the only choice here. Indeed it would be appropriate to consider a pairing-based identity-based encryption (IBE) alternative to the PKI based SSL, as M-Pin is itself a pairing-based and identity-based protocol. We would suggest for consideration the IBE protocol of Sakai and Kasahara [15]. This protocol requires no pairing calculation on the client side, and a single pairing on the server side. So an implementation of M-Pin already contains all of the important building blocks for implementing IBE using the same BN curve.

8 Digital Signature

The M-Pin multi-factor authentication scheme described above can easily be converted in a standard way to a multi-factor digital signature scheme [7] by replacing the random challenge y with $y = H(m, U)$, where m is the message to be signed. The signature then is the tuple $\{U, V\}$, and the server-side authentication step becomes the signature verification. Note this is digital signature in the context where the verification capability is restricted to the entity in possession of the secret verification key sQ. In the literature this is known as a "Strong Designated Verifier Signature" or SDVS [12]. Whereas a regular digital signature can be verified by anyone using the public key associated with the signer, an SDVS can only be verified by the "designated verifier", in this case the M-Pin server equipped with its server secret. Many methods have been proposed for SDVS, but most sacrifice non-repudiation, as the designated verifier can forge signatures. This also means that the designated verifier cannot convince a third party that Alice's signature is valid, because they may have generated it themselves. However the M-Pin SDVS does support non-repudiation as an M-Pin server secret cannot be used to forge Alice's signature. If it ever becomes an issue the M-Pin server secret can be revealed to a judge who can then be convinced that the signature is genuine.

This same idea can be exploited to convert M-Pin from a 3-pass protocol to a 1-pass protocol. Here the Client itself derives the challenge y as $y = H(T, U)$ (where T is a time-stamp now transmitted by the Client along with ID_a, U and V) and the Server checks the accuracy of the time-stamp before completing the protocol. We point out that this 1-pass variant is probably a better choice if M-Pin is to replace an existing Username/Password implementation.

9 Time Permits

Time permits provide a simple alternate revocation capability. The idea is that the server includes an explicitly described time slot in its construction of Alice's hashed identity. Unless Alice has a corresponding "Time permit" for the same time slot, she cannot complete the protocol.

In the protocol above we instead calculate $H(ID_a) + H_T(T_i|ID_a)$ on both sides of the protocol where T_i is a textual description of the i-th time slot and $H_T(.)$ is a hash function distinct from $H(.)$. For the protocol to work correctly Alice must be issued by the Trusted Authority with a permit $s.H_T(T_i|ID_a)$ which gets added to her combined PIN-plus-token secret $s.H(ID_a)$.

Observe that the permit is of no use to any other party, and hence can be issued publicly, or simply sent to Alice by email, or delivered via the server. A proof of security for this idea in the context of Boneh and Franklin IBE can be found in [19].

Alice - identity ID_a	**Server**
Generates random $x < q$	Generates random $y < q$
$A = H(ID_a)$	
$T = H_T(T_i\|ID_a)$	
$D = A + T$	
$U = xD$	
$W = xA$	
$ID_a, U, W \quad \rightarrow$	
	$\leftarrow y$
$V = -(x+y)(((s-\alpha)A + \alpha A) + sT) \rightarrow$	
	$D = H(ID_a) + H_T(T_i\|ID_a)$
	$g = e(V,Q).e(U+yD, sQ)$
	if $g \neq 1$, reject the connection

Table 2. M-Pin with Time Permits

The M-Pin protocol needs a small modification to continue to support the mechanism for determining PIN error. Since this error will be reflected in the PIN-plus-token component and not in the Time Permit, the client must also send the server $W = x.H(ID_a)$ in the first pass of the protocol. Now the value of g returned by the protocol is $g = e(W + yA, Q)^{\delta}$, where δ is the PIN error. See Table 2.

10 Implementation

It is appreciated that for widespread deployment the client side of this protocol might be required to be implemented in the Javascript language inside of a browser, ideally even inside of a browser implemented on a low powered mobile device. On the face of it this is a daunting prospect. However Javascript engines have been improving radically over the last few years, as has the processing power of mobile devices.

Examining the client side of the M-Pin protocol we see that it requires two point multiplications in $\mathbb{G}_1$, These point multiplications in $\mathbb{G}_1$ can benefit from an efficient endomorphism that exists on BN curves, as described by Gallant, Lambert and Vanstone [11]. This makes them roughly twice as fast.

For lap-tops and desk-tops the client-side timings are imperceptible. Some timings for mobile devices are shown in Table 3. On the server side, on contemporary Intel processors, the processing time is of the order of a few milliseconds per connection.

Manufacturer	Model	OS	Browser	M-Pin
Apple	iPhone 4	iOS 6	Safari	2.5
Apple	iPhone 5	iOS 6	Safari	1
Apple	iPad 2	iOS 6	Safari	1
Samsung	Galaxy S3	Android	Chrome	1
Motorola	MB256	Android	Firefox	3

Table 3. M-Pin Client side timings (in seconds)

References

1. L. Ballard, M. Green, B. de Medeiros, and F. Montrose. Correlation-resistant storage via keyword-searchable encryption. Cryptology ePrint Archive, Report 2005/417, 2005. `http://eprint.iacr.org/2005/417`.
2. F. Bao, R. Deng, and H. Zhu. Variations of Diffie-Hellman problem. In *ICICS 2003*, volume 2836 of *Lecture Notes in Computer Science*, pages 301–312. Springer-Verlag, 2003.
3. P.S.L.M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptology – SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer-Verlag, 2006.
4. 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.
5. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.
6. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Asiacrypt 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer-Verlag, 2001.
7. J. Cha and J. Cheon. An Identity-Based signature from gap Diffie-Hellman groups. In *PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 18–30. Springer-Verlag, 2003.

8. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Crypto 1986*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987.
9. D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing friendly elliptic curves. *Journal of Cryptography*, 23:224–280, 2010.
10. S. Galbraith, K. Paterson, and N. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156:3113–3121, 2008.
11. R. Gallant, R. Lambert, and S. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphism. In *Crypto 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer-Verlag, 2001.
12. M. Jakobsson, K. Sato, and R. Impagliazzo. Designated verifier proofs and their applications. In *Eurocrypt 1996*, volume 1040 of *Lecture Notes in Computer Science*, pages 142–154. Springer-Verlag, 1997.
13. 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.
14. J. Pollard. Monte carlo methods for index computation mod p. *Mathematics of Computation*, 32, 1978.
15. R. Sakai and M. Kasahara. ID based cryptosystems with pairing on elliptic curve. Cryptology ePrint Archive, Report 2003/054, 2003. `http://eprint.iacr.org/2003/054`.
16. 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`.
17. M. Scott. Computing the Tate pairing. In *CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304. Springer-Verlag, 2005.
18. N. Smart and F. Vercauteren. On computable isomorphisms in efficient pairing-based systems. *Discrete Applied Mathematics*, 155:538–547, 2007.
19. Y. Tseng and T. Tsai. Efficient revocable ID-based encryption with a public channel. *The Computer Journal*, 55(4):475–486, 2012.
20. X. Yi. An identity-based signature scheme from Weil pairing. *IEEE Communications Letters*, 7:76–78, 2003.