Client-Server Authentication Using Pairings

Michael Scott

Certivox Labs
mike.scott@certivox.com

Abstract. What would be the ideal attributes of a client-server authentication scheme? One might like an identity based scheme not requiring PKI, plus support for multi-factor authentication based on a token, a PIN number, and optionally a biometric. The former might hold a high-entropy secret, and the latter may be represented as relatively low-entropy parameters. However it would be preferred if the token could be in the form of a relatively inexpensive USB stick rather than a Smart-Card. The user should be at complete liberty to choose and change a PIN number, but if they forget it a recovery mechanism should be available. A fuzzy biometric measurement could be supported and accepted if accurate within certain limits. However the Server should not be required to store any information derived from client secrets, so there should be no equivalent of a vulnerable "password file". In fact neither the PIN nor the biometric should be stored anywhere (other than in the client's brain or as part of the client's body respectively). Reasonable performance on relatively low-powered devices should be possible. The damage caused by compromise of the Server and the loss of its long term secrets should be mitigated as much as possible. The property of Perfect Forward Secrecy, a requirement for clients concerned about long-term privacy, should be supported. In this paper we aim to deliver such a scheme.

1 Introduction

A Client-Server authentication protocol should ideally authenticate the Client to the Server, and the Server to the Client, and should result in both of them negotiating strong encryption keys suitable for further communication. Note that this can be regarded as a specialised subset of the more general problem of peer-to-peer authenticated key exchange. However it is in a many-to-one setting, where the roles of clients and servers are quite different and clearly understood. Authenticated Key Agreement (AKE) is one of the classic problems to be solved using modern methods of Cryptography. Identity based methods for key exchange have a long history, going back to an original proposal by Okamoto in the 1980s [12]. In the client-server setting we are particularly interested in techniques that support a multi-factor authentication, that is the client authenticates via a cryptographically strong secret, associated with their identity, and which is divided between a physical token and a memorised PIN number, and (optionally) a biometric measurement.

Such schemes have often been suggested in the past, but they are notoriously difficult to get right, and were often got badly wrong. A classic example would be the scheme proposed by Kim et al. [10], which promised an "ID-based password Authentication Scheme using Smart Cards and Fingerprints". Scott [15] showed how it could be comprehensively broken by an attacker who passively eavesdropped a single transaction. Many other schemes have fallen into the break-fix-break-fix cycle which generates a lot of literature, but rather less confidence.

For some recent papers and a review of some recent literature, see [8] and [18].

To contextualize the problem let us consider it in the setting of an e-banking application.

Its 2016 and the Bank of Oceania rolls out its new e-banking service, wanting to avoid relying on the SSL protocol, and looking for a cheap, secure and user-friendly system. The customer gets two USB stick-like devices. They are identical, and one is to be stored safely in case the original is lost. New ones will be issued every year. Each stick contains a secret unique to the customer and calculated directly from their identity. On inserting the stick into their home computer for the first time, an on-board program fires up and asks the customer to select a PIN number, and, optionally, to enter a biometric. The biometric might be based on immutables like eyes/ears/nose, or on a voice print.

The PIN and the biometric are then subtracted from the secret stored on the stick. When the stick is used for legitimate e-banking, these values are extracted from the client and added back, but only for the duration of the authentication protocol. The stick has its own processing capabilities, and runs its own secure browser to connect to the bank. The user can change their own PIN number locally at any time. For the user the e-banking experience is as simple and familiar as using an ATM.

Now lets try and deliver such a system. We will develop our scheme in the context of two-factor authentication based on a token and a memorised PIN, although the extension to include a fuzzy biometric is obvious.

The property of Perfect Forward Secrecy (PFS) according to its original definition implies that if all of the long term secrets employed in a protocol are compromised, previously recorded transcripts are still secure. Recent authors have rather muddied the water by considering degrees of Perfect Forward Security, depending on exactly which secrets are lost [16]. We take a purist view. The original ephemeral Diffie-Hellman algorithm [11] would be the classic example of a protocol which has the property of PFS. Indeed, it has no long-term secrets. One simple way to enhance a protocol to obtain PFS is to introduce into it a Diffie-Hellman component.

One particularly deadly attack on such schemes is the "off-line dictionary attack". Here an attacker typically captures the token and enough other information (perhaps by eavesdropping a transaction), to go back to his own computer and quickly search through all possible PIN numbers until he finds the unique PIN consistent with the captured information. To this end we must assume that

an attacker has access to some ciphertext encrypted with the finally agreed key, which when decrypted with the correct key results in something instantly recognisable. In other words, an attacker will know when they have hit on the right key.

To avoid this kind of attack the protocol designer must ensure that all information leaked or otherwise accessible to an attacker is consistent with any possible PIN number.

Our notation may be considered a little sloppy. For example a one-way hash function is commonly denoted as simply $H(.)$, where its input and output format are assumed to be clear from the context. The mathematical outcome of the protocols is a mutual key k, but in a practical protocol this is often combined with transmitted values via a hash function to form the final key K.

There is often an implicit assumption when describing key exchange protocols, that received elements are of the correct form and the correct order. For example a point on an elliptic curve is (a) assumed to be on the curve, and (b) assumed to have the expected large prime order. In practice checks should always be made to ensure that these assumptions are true, as failure to do so may open the door to small subgroup confinement attacks [14]. If the checks fail, the protocol should be immediately aborted. In our descriptions below in the interests of clarity, we have omitted these steps.

2 A Poor Man's protocol

One extremely simple solution is that the server maintains a single master secret s, and calculates a shared secret with every client with Identity W of the form $H(W|s)$, where $H(.)$ is a hash function like SHA256. This is simple, fast, robust, and no interaction is required. Every participant has just one secret to protect. In particular the Server does not have to maintain a password file. Distribution of the shared secret is carried out at registration time. It might be considered possible to extract part of the hash as a PIN which would need to be re-inserted in order to reconstruct the shared secret. However an attacker who (a) captures the token, and (b) eavesdrops a packet encrypted with the whole key, can perform an offline dictionary attack to find the PIN by re-inserting every possible PIN until the packet decrypts to something sensible. Furthermore the system is not Perfect Forward Secure (PFS), and the loss of s is of course absolutely fatal.

3 Poor Man redux

Two different approaches are suggested to include a Diffie-Hellman component into the original protocol.

3.1 Embedded DH

What if the original Poor Man's protocol is used instead to establish a secret generator g of a large prime order group, used then as the basis for a Diffie-Hellman key exchange (in the spirit of SPEKE [9])? This supports PFS, and

now a PIN α can apparently be safely extracted from g to create a token. This protocol should be used with a safe prime modulus p of size at least 1024 bits and q at least 160 bits.

Alice - identity ID_a	**Server**
Generates random even $x < q$	Generates random even $y < q$
$ID_a \rightarrow$	
$P_a = ((H(ID_a\|s) - \alpha) + \alpha)^x$	$P_s = H(ID_a\|s)^y$
$P_a \rightarrow$	$\leftarrow P_s$
$K = H(ID_a\|P_a\|P_s\|{P_s}^x)$	$K = H(ID_a\|P_a\|P_s\|{P_a}^y)$

3.2 In Tandem DH

An alternative approach is to simply implement DH in tandem with the original protocol, and combine the contributions from both techniques using a hash function.

Alice - identity ID_a	**Server**
Generates random $x < q$	Generates random $y < q$
$ID_a \rightarrow$	
$P_a = g^x$	$P_s = g^y$
$P_a \rightarrow$	$\leftarrow P_s$
$K = H((H(ID_a\|s) - \alpha) + \alpha\|{P_s}^x)$	$K = H(H(ID_a\|s)\|{P_a}^y)$
$M = H(ID_a\|P_a\|P_s\|K)$	$N = H(ID_a\|P_a\|P_s\|K)$
$M \rightarrow$	if $M \neq N$, drop connection

However if Charlie steals the token but doesn't know the PIN, and if the Server were to respond with something encrypted with the correct key, Charlie can simply try every possible PIN off-line until he hits on the right one. But since there is no ambiguity with respect to the roles of Client and Server we can avoid this problem by insisting that the Client first sends something calculated from the key to the Server first, as shown above. If it does not check out, the Server immediately drops the connection.

Note that this protocol has a key correction capability. For example if the client inputs a PIN number wrong by an amount δ, the Server can easily detect the value of δ, by simply searching through neighbouring keys using $H(ID_a|s) \pm i$ for small values of i until it finds the correct one. As we will see this might be considered as a useful feature. Note that the PIN itself is not revealed to the Server by this process.

However the overall method has a fatal problem. An attacker who captures a token can launch a trivial man-in-the-middle attack which first pretends to be a server to the client, and then launches an off-line dictionary attack to find the PIN, and then uses the stolen token and the recovered PIN to log onto the genuine Server. The details are left as a simple excercise to the reader. The fact

that this attack succeeds might be put down to the fact that a stolen client secret (all or the major part of it) allows the attacker to masquerade as both the client to the Server, and the Server to the client. But if this problem could be fixed...?

A problem with these simple hash-based protocols is that the secret s is used for both enrollment in the system (in the issuing of keys to client), and is also required on-line by the Server. Its loss resulting from an attack on the Server is clearly catastrophic. A better idea is to split the role of the Server from the enrollment role, which should instead be assigned to a trusted third party, which is trusted by both the clients and the Server.

4 Okamoto's protocol

This protocol [12] is not specialised to client-Server, but is worthy of study in that it demonstrates that the possibility of PIN extraction from a secret cannot be assumed. This scheme was one of the first "identity-based" schemes ever to be proposed. Here an individual's identity is offered to the other participant at the start of the protocol, and unless that identity has been properly endorsed by a Trusted Authority (TA), the key exchange will not succeed. This TA issues simple RSA signatures on the identities of each participant. Note that once all such secrets have been distributed the RSA private key can be deleted (in theory) if considered desirable. So identity W is issued with the secret $W^{1/3} \bmod n$, where n is the TAs public key. All arithmetic is modulo n. The key exchange proceeds as follows

Alice - identity ID_a	**Bob - identity** ID_b
Generates random $x < n$	Generates random $y < n$
$ID_a \rightarrow$	$\leftarrow ID_b$
$P_a = H(ID_a)^{1/3}.g^x$	$P_b = H(ID_b)^{1/3}.g^y$
$P_a \rightarrow$	$\leftarrow P_b$
$k = ({P_b}^3/H(ID_b))^x = g^{3xy}$	$k = ({P_a}^3/H(ID_a))^y = g^{3xy}$

Note however that a PIN cannot be extracted from the individual secrets. An attacker who captures the token can simply try every PIN off-line and cube modulo n until they recover the correct ID. The problem is that the TA's public key is known to all, and can be used to check on the validity of any signature.

5 Pairings and PINs

With the advent of pairings other solutions became possible. The main significant property of pairings is that of bilinearity

$$e(aA, bB) = e(bA, aB) = e(A, B)^{ab}$$

where A and B are points on a special pairing-friendly elliptic curve [6].

Here we assume the type-3 pairing [7] where $C = e(A, B)$ is $G_1 \times G_2 \rightarrow G_T$, where $A \in G_1$, $B \in G_2$, and $C \in G_T$. Note that elements from these groups cannot be mixed. We can assume that the pairing-friendly elliptic curve supports such groups of the same large prime order q.

Assume again the existence of an independent TA with its own master secret(s) that is not required on-line – it is only responsible for off-line enrollment and issuing of client and Server ID-based secrets. This provides an extra layer of security and limits the damage caused by the loss of client or Server long-term secrets.

Assume ID_a and ID_s are Alice's identity and the Server's identity respectively. $H_1(.)$ is a hash function that hashes to a point of order q in the group G_1, $H_2(.)$ is a hash function that hashes to a point of order q in the group G_2. Then both the client Alice and the Server are issued with secrets sA and sS respectively, where $A = H_1(ID_a)$, $S = H_2(ID_s)$ and s is the TA's master secret for use with a particular Server. Of course knowing A and sA does not reveal s, as it is protected by a difficult discrete logarithm problem.

Now the simple linear form of these secrets opens up interesting possibilities. Firstly it immediately supports a simple but perfect secret-sharing strategy. A secret sA can be trivially split into two parts s_1A and s_2A where $s = s_1 + s_2$. Furthermore a secret has no special format that makes it identifiable. Any multiple of A is potentially a valid secret, and since A is a generator in G_1, that means that any member of G_1 could be a valid secret.

In practise Alice can extract a PIN α of her choosing from her secret, to divide it into the token part $((s - \alpha)A$, and the PIN part αA. Clearly, when these are added together the full secret can be reconstituted. A captured token is compatible with any possible PIN, and so useless without it.

A very desirable feature is that a rogue client who steals another clients long-term secret should not be able to determine their PIN by performing off-line key-exchanges with themselves. That is, clients should only be able to exchange keys with a Server, not with other clients. By using a type-3 pairing where clients are in G_1 and the Server is in G_2, we achieve this [14].

Note that PIN extraction cannot be used with many pairing-based protocols, for example, Boneh and Franklin's IBE scheme [3], or the Chen and Kudla authenticated Key exchange [5], or any scheme which requires as part of its public parameters a generator point P and $P_{pub} = sP$ in G_2 (or any scheme implemented on a type-1 pairing). In such a case if Charlie were to capture Alice's token containing $(s - \alpha)A$, Charlie could quickly find her PIN by testing all i until

$$e((s - \alpha)A + iA, P) = e(A, sP)$$

This is another example of the off-line dictionary attack, and it is quite deadly. Bilinearity can be as effective an enemy as a friend!

If a Server secret sS is ever leaked (that is revealing S and sS), or indeed any multiple of a known point by s in G_2, then those values can be used to determine the PIN associated with a stolen token. It is of course only common sense that

this should be possible. If the Server secret is discovered, the discoverer can set up their own false Server, and try every possible PIN against a stolen token, and thus discover the PIN.

Note that all clients who access any Server validated by the same TA using the same s, are at risk if just one of those servers in compromised. For this reason each individual server should ideally be associated with a different s master secret.

Interestingly we can deliberately create a circumstance where the Server can launch an off-line dictionary attack on the PIN, and exploit it as a useful feature.

A Server offers a simple PIN-recovery service to the client. The client A sends $X = H(e((s-\alpha)A, S))$ to the Server S, which requires only the token. Then the client presumably goes to some lengths to prove their identity (mother's maiden name etc.). Once that is done to the Server's satisfaction, the Server goes off-line and calculates $Y = H(e(A, sS - iS))$ for all possible i until he gets a match $X = Y$. Then $i = \alpha$. This can then be sent back to the client by email, or some other method.

5.1 Scott's Protocol

A client-server protocol of this type was first proposed by Scott [14]. This paper suggested that whereas a type-1 pairing was suitable for peer-to-peer Authenticated Key Exchange, a type-3 pairing was ideal for the client-Server context. The protocol does not require a TA public key and so PINs can be safely extracted as described above (unlike for Okamoto's protocol).

Alice - identity ID_a	**Server - identity** ID_s
Generates random $x < q$	Generates random $y < q$
$ID_a \rightarrow$	$\leftarrow ID_s$
$S = H_2(ID_s)$, $A = H_1(ID_a)$	$A = H_1(ID_a)$, $S = H_2(ID_s)$
$P_a = e((s-\alpha)A + \alpha A, S)^x$	$P_s = e(A, sS)^y$
$P_a \rightarrow$	$\leftarrow P_s$
$k = {P_s}^x = e(A, S)^{sxy}$	$k = {P_a}^y = e(A, S)^{sxy}$
$K = H(ID_a \vert ID_s \vert P_a \vert P_s \vert k)$	$K = H(ID_a \vert ID_s \vert P_a \vert P_s \vert k)$

This algorithm can be regarded as being derived from the embedded DH approach, as using the SOK non-interactive key exchange algorithm [13] to construct the generator g for use in a Diffie-Hellman key exchange, again rather in the spirit of SPEKE [9]. Therefore this protocol inherits the property of Perfect Forward Secrecy from Diffie-Hellman.

A serious concern is vulnerability to a "Key Compromise Impersonation" (KCI) attack [2], [4]. If Charlie captures Alice's private key and her PIN number, Charlie can not only pretend to be Alice to the Server (which is only to be expected), but Charlie can also pretend to be the Server to Alice. The attack is trivial. Having captured sA Charlie too can calculate $P_s = e(sA, S)^y$. More

seriously if someone should hack the server and extract its secret sS, they could log into that Server claiming any identity Z by calculating $e(Z, sS)$, which is of course the same as $e(sZ, S)$. Again bilinearity is at the root of the problem.

Observe that in the client-server setting (rather than the Peer-to-Peer setting in which key exchange is usually described) KCI can be broken down into Server-to-Client KCI, where the server can masquerade as a client, or Client-to-Server KCI where the client can masquerade as a server, or mutual KCI where both cases are possible. The point being that there could be a scenario where one or other is possible, but not both.

The Key Compromise Impersonation weakness is clearly not desirable. Fortunately it can be fixed at the cost of some further calculations and bandwidth.

Alice	**Server**
Generates random $x, z < q$	Generates random $y, w < q$
$ID_a \rightarrow$	$\leftarrow ID_s$
$S = H_2(ID_s)$, $A = H_1(ID_a)$	$A = H_1(ID_a)$, $S = H_2(ID_s)$
$P_z = zA \quad \rightarrow$	$\leftarrow P_w = wS$
$P_a = e((s-\alpha)A + \alpha A, P_w + zS)^x$	$P_s = e(P_z + wA, sS)^y$
$P_a \rightarrow$	$\leftarrow P_s$
$K = H(ID_a \vert ID_s \vert P_z \vert P_w \vert P_a \vert P_s \vert P_s^x)$	$K = H(ID_a \vert ID_s \vert P_z \vert P_w \vert P_a \vert P_s \vert P_a^y)$

The mutual key is derived from $k = e(A, S)^{sxy(w+z)}$. Next is another modification of Scott's Protocol which prevents Server-to-Client KCI, but not Client-to-Server KCI, and keeps client-side calculations to a minimum.

Alice	**Server**
Generates random $x < q$	Generates random $y, w < q$
$ID_a \rightarrow$	$\leftarrow ID_s$
$S = H_2(ID_s)$, $A = H_1(ID_a)$	$A = H_1(ID_a)$, $S = H_2(ID_s)$
	$\leftarrow P_w = wS$
$P_a = e((s-\alpha)A + \alpha A, P_w + S)^x$	$P_s = e(A + wA, sS)^y$
$P_a \rightarrow$	$\leftarrow P_s$
$K = H(ID_a \vert ID_s \vert P_w \vert P_a \vert P_s \vert P_s^x)$	$K = H(ID_a \vert ID_s \vert P_w \vert P_a \vert P_s \vert P_a^y)$

This time the mutual key is derived from $k = e(A, S)^{sxy(w+1)}$.

6 Wang's protocol.

Scott's method is not the only pairing-based protocol that can support PIN extraction. Here we adapt Wang's protocol [17] for the client-Server setting, and base it on a type-3 pairing. In this setting this protocol has the same desirable properties as Scott's (except that it does not immediately have the property of Perfectly Forward Secrecy), and it is arguably more efficient to implement.

Furthermore Wang's protocol has a formal proof of security and forms part of the P1363.3 proposed IEEE standard [1].

We start with a much simplified version of Wang's protocol.

Alice - identity ID_a	**Server - identity** ID_s
Generates random $x < q$	Generates random $y < q$
$ID_a \rightarrow$	$\leftarrow ID_s$
$S = H_2(ID_s)$, $A = H_1(ID_a)$	$A = H_1(ID_a)$, $S = H_2(ID_s)$
$P_a = xA$	$P_s = yS$
$P_a \rightarrow$	$\leftarrow P_s$
$k = e(x.((s-\alpha)A + \alpha A), P_s) = e(A,S)^{sxy}$	$k = e(P_a, y.sS) = e(A,S)^{sxy}$

Wang's protocol is not vulnerable to a Key Compromise Impersonation attack. However the TA who has eavesdropped on P_a and P_s can easily calculate the key as $e(P_a, P_s)^s$. This is as a direct result of the protocol not having the property of Perfect Forward Secrecy (PFS).

To include the property of PFS, Wang suggests this modification which includes an in-tandem Diffie-Hellman into the key exchange. See also [5].

Alice - identity ID_a	**Server - identity** ID_s
Generates random $x < q$	Generates random $y, w < q$
$ID_a \rightarrow$	$\leftarrow ID_s$
$S = H_2(ID_s)$, $A = H_1(ID_a)$	$A = H_1(ID_a)$, $S = H_2(ID_s)$
$P_a = xA$	$P_s = yS, P_g = wA$
$P_a \rightarrow$	$\leftarrow P_s, P_g$
$k = e(x.((s-\alpha)A + \alpha A), P_s)$	$k = e(P_a, y.sS)$
$K = H(ID_a \vert ID_s \vert P_a \vert P_s \vert P_g \vert xP_g \vert k)$	$K = H(ID_a \vert ID_s \vert P_a \vert P_s \vert P_g \vert wP_a \vert k)$

Both keys are the same $K = H(ID_a|ID_s|P_a|P_s|P_g|xwA|e(A,S)^{sxy})$. In Wang's original paper it is suggested to use $w = y$. However this enables the Key Compromise Impersonation attack, based on the observation that $e(xsZ, yS) = e(xyZ, sS)$, so if sS is captured, Charlie can log in to S with any identity Z.

We now make an interesting observation about Wang's protocol and protocols like it. If the client's reconstituted secret is off by a small amount, a mutual key can still be calculated by the Server, who can determine the extent of the error and compensate for it. For example if the client uses $sA + iA$ as their secret, the Server can compensate by using $sS + iS$ as its secret. Furthermore the Server can afford to spend some time searching for the right i. Therefore this scheme supports a key correction capability, but this time because of its KCI resistance it is not vulnerable to a man-in-the-middle attack.

7 PIN issues

The PIN number extraction idea needs to be used with extreme care in the case of Wang's protocol. Lets say Charlie steals Alices token but does not know her PIN number α. Charlie then attempts a connection to the Server. Charlie enters into the protocol but calculates a key just using the token value, as $C = e(A,S)^{(s-\alpha)xy}$. The Server behaves properly and calculates the key using $k = e(A,S)^{sxy}$.

Now during the protocol Charlie and the Server swap xA and yS. So Charlie can calculate $W = e(xA, yS) = e(A,S)^{xy}$. Next the Server sends something to Charlie encrypted using k. At this stage Charlie drops the connection, and offline keeps attempting decryption using CW^i until he gets something sensible, at which stage $i = \alpha$, and he has recovered Alice's PIN number. Because when $i = \alpha$

$$e(A,S)^{(s-\alpha)xy} e(A,S)^{ixy} = e(A,S)^{sxy}$$

A simpler, but slower, way is for Charlie to simply calculate the key using all possible PINs until he finds one that decrypts properly.

This off-line dictionary attack can be defeated by again insisting in the protocol that Alice send something encrypted with the key she has just calculated, before the Server sends anything to Alice. If the Server fails to decrypt it using the correct key, the Server must immediately drop the connection. Such an enforced ordering of messages is common in client-server protocols [19].

This potential attack does not apply to Scott's protocol.

The same behavior can be exploited in a positive way. If the client were to enter the wrong PIN, the server can easily determine the extent of the error δ using the key correction capability. So for example if the client entered 1224 instead of 1234, then the server could figure out that the PIN was "out" by 10. This is again because of bilinearity.

$$e((s+\delta)A, P_s) = e(P_a, y.(sS + \delta S)) = e(P_a, y.sS).e(P_a, yS)^{\delta}$$

Basically the error on the client side can be cancelled by invoking the same error on the Server side. The Server can calculate $W = e(P_a, yS)$ and keep trying $k.W^i$ for small values of i until they find the correct key (which decrypts something sent by the client to something sensible).

This behaviour can be exploited to intelligently respond to the wrong PIN being entered – if it is out by a lot, do not allow another attempt, if it is out by only one digit, then allow a further attempt, etc. A "coercion" convention could be agreed, for example if the PIN was just out by 1 in the last digit, the Server might interpret this as a client entering a PIN while under physical threat. In which case the Server might respond normally - but call the cops!

More interestingly this behaviour might be exploited to support a low entropy biometric feature. A biometric is a naturally fuzzy measurement which may well be out by a small amount while still being acceptable. Since the Server

can determine the extent of the error, an informed decision can be made as to whether the biometric is "close enough" to be accepted.

8 A Two Factor Client-Server Protocol

The key correction capability makes us tend to favour the Wang protocol as the basis of our proposed scheme, even though it is more complex, and care must be taken to get it right.

As we have described it above Wang's protocol has a fatal problem. The identities of A (and S) are not directly included in the key calculation. So Bob can claim the identity Alice, but still log on using his own credentials. This is clearly not satisfactory. Wang uses a rather complex method to fix this, as we will see. His solution has the useful side effect of ensuring that the ephemeral public keys (P_a, P_s and P_g above) cannot be tampered with in transit.

Assume $H_q(.)$ is a hash function that hashes to a number in the range 1 to q (although according to Wang its OK to reduce this range to a size half the number of bits in q, with some performance gains).

Alice	**Server**
Generates random $x < q$	Generates random $y, w < q$
$ID_a \rightarrow$	$\leftarrow ID_s$
$S = H_2(ID_s)$, $A = H_1(ID_a)$	$A = H_1(ID_a)$, $S = H_2(ID_s)$
$P_a = xA \quad \rightarrow$	$\leftarrow P_s = yS, P_g = wA$
$r_a = H_q(P_a\|P_s\|P_g)$, $r_s = H_q(P_s\|P_a\|P_g)$	$r_s = H_q(P_s\|P_a\|P_g)$, $r_a = H_q(P_a\|P_s\|P_g)$
$k = e((x + r_a)((s - \alpha)A + \alpha A), r_s S + P_s)$	$k = e(r_a A + P_a, (y + r_s)sS)$
$K = H(k\|xP_g)$	$K = H(k\|wP_a)$
$M = H(ID_a, ID_s, K)$	$N = H(ID_a, ID_s, K)$
$M \rightarrow$	if $M \neq N$, drop the connection

For both parties observe that $k = e(A, S)^{s(x+r_a)(y+r_s)}$. Observe (and take comfort) from the fact that Alice's token and PIN are recombined locally before any value calculated from them is transmitted, so no-one is in the position to determine the PIN from transmitted values, irrespective of their computing power. If the wrong PIN is entered, the Server drops the connection (and only allows a few more attempts before taking more drastic action with respect to the purported "Alice").

9 Conclusions

Note that Wang's protocol is not the only choice here. There is for example an alternative protocol due to Wang et al. [16] which also supports a PIN and key correction. It is rather more elegant than Wang's, and claims to be faster. It also has a nice and simple proof of security.

Finally consider the implications of a successful hack on the Server which reveals sS. Note that there are no client related secrets (token, PIN or biometric) stored on the Server. However this attack obviously allows a false server to be set up, onto which clients might be convinced to log on. If the successful hacker then captures a token, they can easily find the associated PIN. However they cannot use the captured Server secret to log onto the genuine Server, and access its data. They cannot enroll any other clients. So although loss of the Server secret is bad, the effects are to an extent mitigated. Of course loss of the TA master secret s is catastrophic, except to the extent of protection offered by Perfect Forward Secrecy.

References

1. IEEE P1363 home page. `http://grouper.ieee.org/groups/1363/`.
2. S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. *Cryptography and Coding*, 1355:30–45, 1997.
3. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.
4. L. Chen, Z. Cheng, and N. P. Smart. Identity-based key agreement protocols from pairings. *Int. J. Inf. Secur.*, 6:213–241, June 2007.
5. L. Chen and C. Kudla. Identity based key agreement protocols from pairings. In *Proc. of the 16-th IEEE Computer Security Foundations Workshop*, pages 219–213. IEEE Computer Society, 2002.
6. D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing friendly elliptic curves. *Journal of Cryptography*, 23:224–280, 2010.
7. S. Galbraith, K. Paterson, and N. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156:3113–3121, 2008.
8. F. Hao and D. Clarke. Security analysis of a multi-factor authenticated key exchange protocol. Cryptology ePrint Archive, Report 2012/039, 2012. `http://eprint.iacr.org/2012/039`.
9. D. Jablon. Strong password-only authenticated key exchange. *Computer Communication Review (ACM SIGCOMM)*, 26(5):5–26, 1996.
10. H. S. Kim, S. W. Lee, and K. Y. Yoo. ID-based password authentication scheme using smart cards and fingerprints. *ACM Operating Systems Review*, 37(4):32–41, 2003.
11. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. CRC Press, Boca Raton, Florida, 1996.
12. E. Okamoto. Proposal for identity-based key distribution systems. *Electron. Lett.*, pages 1283–1284, 1986.
13. R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan, 2000.
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. M. Scott. Cryptanalysis of an ID-based password authentication scheme using smart cards and fingerprints. Cryptology ePrint Archive, Report 2004/017, 2004. `http://eprint.iacr.org/2004/017`.

16. Shengbao Wang, Zhenfu Cao, Zhaohui Cheng, and Kim-Kwang Raymond Choo. Perfect forward secure identity-based authenticated key agreement protocol in the escrow mode. *Science in China Series F Information Sciences*, 52(8):1358–1370, 2009.
17. Y. Wang. Efficient identity-based and authenticated key agreement protocol. Cryptology ePrint Archive, Report 2005/108, 2005. `http://eprint.iacr.org/2005/108`.
18. Y. Wang. Password protected smart card and memory stick authentication against off-line dictionary attacks. Cryptology ePrint Archive, Report 2012/120, 2012. `http://eprint.iacr.org/2012/120`.
19. T. Wu. The secure remote password protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, 1998.