Password Authentication Using Multiple Servers

David P. Jablon

Integrity Sciences, Inc.
www.IntegritySciences.com
dpj@world.std.com

Abstract. Safe long-term storage of user private keys is a problem in client/server systems. The problem can be addressed with a roaming system that retrieves keys on demand from remote credential servers, using password authentication protocols that prevent password guessing attacks from the network. Ford and Kaliski's methods [11] use multiple servers to further prevent guessing attacks by an enemy that compromises all but one server. Their methods use a previously authenticated channel which requires client-stored keys and certificates, and may be vulnerable to offline guessing in server spoofing attacks when people must positively identify servers, but don't. We present a multi-server roaming protocol in a simpler model without this need for a prior secure channel. This system requires fewer security assumptions, improves performance with comparable cryptographic assumptions, and better handles human errors in password entry.

1 Introduction

Cryptographic systems that can tolerate human misbehavior are evolving, with fitful progress. A persistent theme is that people tend towards convenient behavior despite well-intentioned security advice to the contrary. It's hard for us to memorize and type strong cryptographic keys, so we use weak passwords. It's hard for us to take the necessary steps to insure that our web browser is securely connected to the correct web server, so we don't. To counter these problems, designers of security systems must accept our weaknesses, and must not assume that we can fully control these human devices. [1, 8, 20]

The common practice of storing password-encrypted private keys in workstation files is a backwards evolutionary step. Long-term storage of password-crackable keys on a poorly managed machine creates opportunity for theft and eventual disclosure of these keys. An ideal system stores password-derived data only in the user's brain and other secure locations, such as a well-managed server, or perhaps a smartcard. When a global network is usually available, but smartcards are not [1], it seems a shame to degrade the power of private keys with persistent untrustworthy storage. Roaming protocols address this problem.

[1] In saying that smartcards are "not available", we mean with card readers on all acceptable machines and cards in all relevant pockets, noting that inconvenience, cost, and other human issues often pose barriers to use and deployment.

This paper describes a new roaming protocol that can use just a small password to securely retrieve and reconstruct a secret key that has been split into shares distributed among multiple servers. The system prevents brute-force attack from an enemy that controls up to all but one of the servers, and has fewer security assumptions, higher performance, and higher tolerance of human misbehavior than similar methods previously described.

The new system does not require prior server-authentication, as does earlier work [11] that relies on techniques like server-authenticated Secure Sockets Layer (SSL) [12, 7], which is known to be vulnerable to web-server spoofing problems. [6, 8] A further advance is to decrease the amount of computation by using smaller groups, without introducing new cryptographic assumptions. Finally, we show how the protocol better tolerates human errors in password entry, by insuring that corrected typographical errors are gracefully ignored and are not counted against the user as suspected illegal access attempts.

These benefits can also be realized in non-roaming configurations.

2 History of Roaming Protocols

The goal of a roaming protocol is to permit mobile users to securely access and use their private keys to perform public-key cryptographic operations. We refer to mobility in a broad sense, encompassing acts of using personal workstation, and other people's workstations, without having to store keys there, using public kiosk terminals, as well as using modern handheld wireless network devices. We want to give users password-authenticated access to private keys from anywhere, while minimizing opportunities for an enemy to steal or crack the password and thereby obtain these keys.

Smartcards have promised to solve the private key storage problem for roaming users, but this solution requires deployment of cards and installation of card readers. The tendency for people to sacrifice security for convenience has proved to be a barrier to widespread use of solutions requiring extra hardware. This is one motivation for software-based roaming protocols.

Throughout the rest of this paper *roaming protocol* refers to a secure password-based protocol for remote retrieval of a private key from one or more credentials servers. Using just an easily memorized password, and no other stored user credentials, the user authenticates to a *credentials server* and retrieves her private key for temporary use on any acceptable client machine. The client uses the key for one or more transactions, and then afterwards, erases the key and any local password-related data.

In our discussion we refer to the user as *Alice* and to credentials servers generally as *Bob*, or individually as B_i, using gender-specific pronouns for our female client and her male servers.

Roaming. The SPX LEAF system [24] presents a roaming protocol that uses a server-authenticated channel to transmit a password to a credentials server for verification, and performs subsequent retrieval and decryption of the user's

private key. The credentials server protects itself by limiting guessing attacks that it can detect, and the protocol prevents unobtrusive guessing of the password off-line.

When a credentials server can determine whether a password guess is correct, it can prevent or delay further exchanges after a preset failure threshold.

Password-Only Protocols. The EKE protocols [1] introduced the concept of a secure *password-only protocol*, by safely authenticating a password over an insecure network with no prior act of server authentication required. A series of other methods with similar goals were developed, including "secret public key" methods [13, 15], SPEKE [18], OKE [21], and others, with a growing body of theoretical work in the password-only model [16, 4, 2, 3]. Most of these papers stress the point that passwords and related memorized secrets must be conservatively presumed to be either crackable by brute-force or, at best, to be of indeterminate entropy, and this warrants extra measures to protect users.

The roaming model and password-only methods were combined in [23] to create protocols based on both EKE and SPEKE. These authors showed that simple forms of password-only methods were sufficient for secure roaming access to credentials. Other roaming protocols were described in [13, 15], [26], [16], and [22], all being designed to stop off-line guessing attacks on network messages, to provide strong software-based protection when client-storage of keys is impractical.

Multi-server Roaming. In a further advance, Ford and Kaliski described methods [11] that use multiple servers to frustrate server-based password cracking attacks to an amazing degree. Single-server password-only protocols prevent guessing attacks from the client and the network but do not stop guessing based on password-verification data that might be stolen from the server. At the cost of using n related credentials servers to authenticate, Ford and Kaliski extended the scope of protection to the credentials server database. In their methods, an enemy can take full control of up to $n-1$ servers, and monitor the operation of these servers during successful credential retrieval with valid users, and still not be able to verify a single guess for anyone's password, without being detected by the remaining uncompromised server.

Yet, the methods detailed in [11] all rely on a prior server-authenticated channel. We believe this is a backwards evolutionary step, in introducing an unnecessary and potentially risky security assumption. We remove this dependency on a prior secure channel for password security, and present other improvements in our description of a new *password-only multi-server roaming protocol* in Section 4.

3 Review of Ford and Kaliski

We review here three methods described in [11], focusing particularly on one which we refer to as FK1.

3.1 FK1

FK1 uses multiple credentials servers, with a splitting technique to create multiple shares of a master key, and a blinding technique to get each share back from a credentials server without revealing the password to the server or anyone else in the process.

The authors used the term *password hardening* to refer to their key-share retrieval process, which seems essentially the same concept as *password amplification* in [1]. To avoid confusion we avoid using either of these terms outside of their original contexts. Also, in our description of FK1, we take some liberties in interpreting their protocol by using a cryptographic hash function h for a few different purposes, and we use a somewhat different notation than in their paper. (Table 1 in Section 4.2 summarizes the notation that we use to describe both FK1 and our methods.)

FK1 Parameters. The FK1 system operates in a subgroup of order q of $\mathbb{Z}_p^*$ where $p = 2q + 1$ with prime p and q. The system uses $n > 1$ credentials servers, and all exponentiation is done modulo p.

FK1 Enrollment. To enroll in the system, the user, Alice, selects a password P, and a series of random numbers $\{y_1, \ldots y_n\}$, each in the range $[1, q-1]$. For each $i \in [1, n]$ she computes a secret key share $S_i := (h(P))^{2y_i}$ using a mask generation function h.

Alice sends each y_i along with her identifier A in an enrollment message to the i^{th} server, B_i. Alice computes a master key K_m for herself using a key derivation function of the shares, $K_m := h(S_1, \ldots, S_n)$. Then, using independent keys derived from K_m, she encrypts some of her most private secrets to be stored wherever she desires.

K_m is clearly a strong secret, as is each share S_i, and a neat result of this construction is that it incorporates P into every share, but it is impossible for an attacker to even verify trial guesses for P, unless he obtains all n shares.

FK1 Authenticated Retrieval. To retrieve her master key at a later time, Alice chooses a random $x \in_R [1, q-1]$, computes $Q := (h(P))^{2x}$, and for each $i \in [1, n]$, she sends Q to B_i. Each B_i computes $R_i := Q^{y_i}$, and sends R_i in reply.

Client: { **request**, A, $Q = h(P)^{2x}$ } $\rightarrow B_i$
Server B_i: { **reply**, $R_i = Q^{y_i}$ } $\rightarrow$ ***Client***

The value x serves as a blinding factor, to insure that the password cannot be determined from Q, and R_i is essentially a blinded form of the key share.

Alice recovers each key share with $S_i := R_i^{1/x \bmod q} = (h(P))^{2y_i}$. which removes the blinding factor. She then reconstructs her master key.

Alice then derives n unique authentication keys (for each $i \in [1, n], K_i := h(K_m||i)$). Although their paper is not specific on how it should be done, Alice uses the K_i derived keys to authenticate to each B_i.

Each server B_i then reconciles each act of authentication with the corresponding received Q value, to determine whether the **request** was legitimate, or perhaps an invalid guess from an attacker.

3.2 Server Pre-authentication

FK1 requires server-authenticated connections to each server to prevent an evil party (who perhaps has compromised one of the servers) from controlling the communications channels to all the servers. The problem is this: An enemy who controls all channels can substitute known false replies, and then perform an attack on the expected value of the combined key as revealed by Alice. Note that the combined key is completely determined by the password and the replies sent by the attacker. If the user reveals information about K_m to an enemy who also knows all y_i, the enemy can verify off-line guesses for P.

The description in [11] does not specify an explicit method for server authentication, but does suggest the use of SSL. It also suggests that server-authentication is optional in the case where the server gives the user "a proof of knowledge that $[R_i]$ was computed from $[Q]$ with the correct exponent." However, no proof or verification process is described.

To establish a secure channel to the server with a typical server-authenticated SSL solution, as implemented in a web browser, requires the client to have a root key for one or more certificate authorities. It also requires the server to have access to a chain of certificates that associate a client root key with the public key of the named server. The client must further include certificate validation software and policy enforcement to validate a certificate of the appropriate server (or servers) selected by the user. All of this is fairly standard. However, ultimately the user must insure that the server name binding is correct. This requires significant action and attention by the user – which the user can easily omit.

The complete reliance on SSL, especially if used in the browser model, is risky. The user can be tricked into using "valid" SSL connections to malicious servers, or tricked into not using SSL at all. This process is subject to several types of failure. While these failures might be called human error, in our view the error is in having unrealistic expectations of the human participant.

Furthermore, if we presume that a common set of root certificates in the client can validate both servers, we've now introduced one or more single points of failure into the system. There is effectively a single point of attack at each location where a private key resides for the root or any subordinate certificate authority. This may be significant, as a primary goal of the multi-server model is to eliminate single points of failure.

The aforementioned certificate chain attack can be achieved by compromising any single system that has a key to create a valid-looking certificate chain for the two servers in question. Furthermore, as described above, an attack in

SSL browser model can trick the user into using "valid" SSL connections to malicious servers, or into not using SSL at all. To counter these threats, in some environments, the identity of the server may be fixed in the configuration of the client, but this approach severely limits functionality.

Our main point regarding this issue is that the risks here, however great or small, are *unnecessary*. We remove the dependency on a prior server-authenticated channel in our alternative model.

3.3 Other Variations – FK2, FK3

Two variations on FK1 that were also presented in [11] include a method using blind signatures [5], and a "special case" method that uses a password-hardening server to convert a password into a key that is suitable for authenticating to a conventional credentials server. We'll call these methods FK2 and FK3, respectively. The authors suggest that the communications channel to the conventional server needs to be integrity protected. In fact, both servers' channels need to be protected.

Handling Bad Access Attempts. In their discussion of FK3, it is suggested that the client authenticate itself with "the user's private key" and that the server keep track of the number of password hardenings and reconcile this with the number of successful authentications. If there are "significantly more" hardenings than authentications, then the account would be locked out.

We note that unsuccessful logins may be quite common. Passwords are frequently mis-typed, and users may often enter the wrong choice of multiple passwords, before finally getting it right. If a long-term fixed limit is place on such mistakes, valid clumsy users might be locked out. On the other hand, if the system tolerates a three-to-one attempt-to-success ratio, an occasional guessing attack by an enemy over the long term might remain undetected.

To address this problem, the system should be *forgiving*, and not account for transient mistakes by a valid user in the same way as invalid access attempts by unknown users. Our protocol addresses this problem. We provide detail for an alternative reconciliation process in our method to deal with transient password-entry mistakes by the user.

4 New Protocol

We now present our improved model for a password-only multi-server roaming protocol, comparing it to model used in FK1, followed by a more detailed description of our protocol.

4.1 New Model

Our model for multi-server roaming is similar to that in FK1, but with some new features and characteristics.

First, our model permits authentication messages to be sent over an unprotected channel; No SSL is required. To prevent the possibility that an enemy in control of the channel can trick Alice into using an improper master key, Alice confirms that the master key is correct before using it to create any data that might be revealed to the enemy.

Second, the authentication step uses a signed message to authenticate valid logins, as well as prior legitimate-but-mistaken logins.

Enrollment Model. At enrollment time, Alice creates n shares of a master symmetric key K_m where each i^{th} share S_i is formed as a function of her password P raised to a random exponent y_i. The shares are combined with a function such that an attacker who has knowledge of any proper subset of shares cannot distinguish K_m from a random value in the same range.

Alice then somehow conveys each exponent y_i to be stored as a closely guarded secret by the i^{th} server.

Alice also selects a public/private key pair $\{V,U\}$ for digital signatures, and symmetrically encrypts private key U using a key derived from K_m to create her encrypted private key U_K. Finally, she creates a proof value $proof_{PK_m}$ that links the password to her master key.

Alice sends V to each of the n servers, and stores U_K and $proof_{PK_m}$ in a convenient place, perhaps on each of the servers. The enroll protocol flow must be performed through a secure channel that authenticates the identity of Alice, A, to each i^{th} server B_i.

Client: { **enroll**, A, V, y_i } $\rightarrow B_i$
Client: { **record**, A, U_K, $proof_{PK_m}$ } $\rightarrow B_i$

Authentication Model. At login time, to reconstitute her master key and retrieve her private key, Alice sends a randomly blinded form of the password Q to each server. Each server in turn responds with a blinded reply R_i consisting of the blinded password raised to power of the secret exponent value ($R_i := Q^{y_i}$) which represents a blinded share of the user's master key. At least one of the server's also sends Alice her encrypted private signature key U_K and $proof_{PK_m}$.

Client: { **request**, Q } $\rightarrow B_i$
Server B_i: { **reply**, Q^{y_i}, U_K, $proof_{PK_m}$ } $\rightarrow$ ***Client***

Interestingly, the channel though which Alice retrieves U_K and $proof_{PK_m}$ does not have to guarantee the integrity of these values. This is discussed further in Section 4.4.

Alice unblinds each reply to obtain each key share and combines the shares to rebuild her master key K_m. She then verifies that the master key is correct using the proof value $proof_{PK_m}$ and her password P. If the proof is incorrect, this implies that at least one of the key shares must be incorrect, and she must abort the protocol without revealing any further information about K_m or P to the

network. Otherwise, she uses a key derived from K_m to decrypt her encrypted private key (and any other desired data), and then completes the protocol by proving her identity to each server. For each blinded password Q that she recently sent to each server, she sends a signed copy of the blinded password.

Client: {**confirm**, Q_1, $\{Q_1\}_U$ } $\rightarrow B_i$
Client: {**confirm**, Q_2, $\{Q_2\}_U$ } $\rightarrow B_i$
...

Each server matches the signed Q_x values from Alice against its list of recently received blinded passwords, and removes any matching entries that are accompanied by valid signatures. The remaining entries, if not confirmed within a reasonable amount of time, are considered to be suspected illegal access attempts, which we label *bad*. Counting *bad* access attempts may be used to limit or delay further blinded share replies for the user's account if the counts rise above certain thresholds.

Verification of Master Key. As mentioned above, one new feature of our method is that Alice can perform the authentication over insecure channels. She retrieves (typically from a credentials server) her verifier $proof_{PK_m}$, and then confirms the validity of the reconstructed master key by comparing a keyed hash of her password with it to $proof_{PK_m}$. If the values don't match, Alice aborts the protocol.

Verification of Legal Access. Another enhancement of our method relates to how Alice proves knowledge of the master key to each server, and how each server reconciles this information with its own record of access attempts.

As in FK1, the servers detect illegal access attempts by looking for a message from Alice that contains a proof of her knowledge of the master key, and by implication, proof that she knows her password. If a valid proof is not associated with the blinded password value, the server must trigger a *bad access event* for Alice's account. Our method differs from FK1 in our detailed description of the construction of Alice's proof and how each server uses the proof to *forgive* Alice's mistakes in password entry.

In FK1, the user authenticates to each server using a unique key derived from the master key. We note that when not using SSL, simply sending $h(K_m||i)$ to B_i would expose the method to a replay attack. To prevent this, we make the proof incorporate the blinded request value that is sent by Alice. Furthermore, we recognize that Alice occasionally mis-types her password, and we'd rather not penalize her by incrementing her illegal access count, which might cause premature account lockout. We want each server to forgive her mistakes, when she can subsequently prove to the server that she ultimately was able to enter the correct password.

Forgiveness Protocol. User's honest mistakes are forgiven by sending evidence of recent prior invalid access attempts after each successful authentication. Upon receiving and validating this evidence, each server erases the mistake from the record, or records the event as a corrected forgivable mistake. By fine-tuning a server's event log in this manner, a system administrator gets a more detailed view of when the system is truly at risk, as opposed to when valid users are merely being frustrated.

A forgiving system seems to require at least one signature generation step on the client and one signature verification step for each of the servers. To minimize computation, the signature steps provide the combined functions of authenticating the user, and proving that the request came from that user. In constructing a valid authentication message for a user, the client includes the set of all recent challenge messages issued by that user, digitally signs the result with the user's private key, and sends it to all servers. Each server verifies the signature to authenticate the user, and at the same time validate evidence of her recent forgivable mistakes.

Each server, upon receiving Alice's **confirm** message, will attempt to reconcile her proof of her access attempts against his recorded list of recent attempts. He does this by verifying Alice's signature on each Q value. Upon successful verification, he knows that the Q value was indeed sent by someone who ultimately knew the password, regardless of whether that request message was specifically used to recreate her master key.

4.2 Detailed Protocol

We now describe an implementation of the protocol in detail, using the notation summarized in Table 1 below.

Parameters. In this protocol we define two security parameters, j which represents the desired bit-strength for symmetric functions, and k representing the number of bits required for the modulus of asymmetric functions.

We define G_q as the subgroup of order q in Z_p^*, where p, q and r are odd primes, $p = 2rq + 1$, $2^k > p > 2^{k-1}$, $r \neq q$, and $2^{2j} > q > 2^{2j-1}$. We also use a function that maps a password to a group element $g_P \in \mathrm{G}_q$, and suggest that $g_P = h(P)^{2r} \bmod p$.

(Alternately one might use an elliptic curve group in $GF(p)$ with a group of points of order $r \cdot q$ approximately equal to p, prime q, and small co-factor $r \in [1, 100]$ or so. In this case we would replace all exponentiation with scalar point multiplication, and define $g_P = r \cdot point(h(P))$, where $point$ uses $h(P)$ to seed a pseudo-random number generator to find an arbitrary point on the curve. [17])

Enrollment. Alice selects a password P, computes $g_P := h(P)^{2r}$, and creates a private key U and corresponding public key V suitable for performing digital signatures.

Table 1. Notation

Symbol	Meaning [Reasonable example]
C_i	list of credentials stored by B_i
g_P	element of G_q corresponding to P $[h(P)^{2r}]$
G_q	group of prime order q
	[in $Z\!\!Z_p^*$, $p = 2rq + 1$, $2^{2j} > q > 2^{2j-1}$, $2^k > p > 2^{k-1}$, p and r prime]
h	a hash function [h = SHA1]
j	security parameter for resisting brute-force attack [80]
k	security parameter for resisting NFS discrete log attack [1024]
K_i	Shared key between Alice and B_i $[h(K_m \| \| i)]$
K_m	Alice's master key, a hash of concatenated shares, $h(S_1 \| \| \ldots \| \| S_n) \bmod 2^j$
L_i	list of suspected bad attempts stored by B_i
P	user's password, $0 < P < 2^{2j}$ [SHA1(password)]
R_i	a blinded key share $= {g_P}^{xy_i}$
S_i	a key share $= {g_P}^{y_i}$
U	Alice's private signing key
U_K	Alice's encrypted private key $= {}_{K_m}\{U\}$
V	Alice's public key corresponding to U
y_i	Alice's secret share exponent stored by B_i
${}_x\{y\}$	message y encrypted with symmetric key x
${}_{1/x}\{y\}$	message y decrypted with symmetric key x
$\{y\}_x$	message y signed with private key x

She then creates n key shares where each i^{th} share $S_i \in G_q$ is formed as $S_i := {g_P}^{y_i}$ using randomly chosen $y_i \in_R [1, q-1]$. She then creates her master j-bit symmetric key with $K_m := h(S_1 \| \| \ldots \| \| S_n) \bmod 2^j$, creates her encrypted private key as $U_K := {}_{K_m}\{U\}$, and creates her key verifier $proof_{PK_m} := h(K_m \| \| g)$.

To enroll these credentials, the client sends Alice's credentials to be stored in a list C_i maintained on each B_i. They must perform these actions using an authenticated communication method that assures the proper identity of A:

Client: for each $i \in [1, n]$, { **enroll**, A, y_i, V, U_K, $proof_{PK_m}$ } $\rightarrow B_i$
Servers: store { A, y_i, V, U_K, $proof_{PK_m}$ } in C_i

Authenticated Retrieval. For authenticated credential retrieval, the client and servers and perform the actions listed below. In this process, each server maintains a list L_i containing a record of suspected bad access attempts.

Client:
select a random number $x \in [1, q-1]$
$Q := {g_P}^x$ modp
{ **request**, A, Q } $\rightarrow$ ***Servers***

Servers:
retrieve { A, y_i, V, U_K, $proof_{PK_m}$ } from C_i

$t := CurrentTime$
append { A, Q, V, t } to L_i
$R_i := Q^{y_i}$
{ **reply**, R_i, U_K, $proof_{PK_m}$ } $\rightarrow$ ***Client***

Client:
for each $i \in [1, n]$,
$S_i := R_i^{1/x} \bmod p$
$K' := h(S_1||S_2||\ldots||S_n)$
if $proof_{PK_m} \neq h(K'||g)$, abort
$U := {}_{1/K'}\{U_K\}$
for Q' in { Q, Q_1, Q_2, $\ldots$ }
{ **confirm**, Q', $\{Q'\}_U$ } $\rightarrow$ ***Servers***}

Servers:
for each received { **confirm**, Q', $\{Q'\}_U$ }
for any $\{A,Q,V,t\}$ in L_i where $Q = Q'$
verify $\{Q'\}_U$ as signature of Q with V
if the signature is valid,
remove $\{A,Q,V,t\}$ from L_i

Periodically, perhaps once a minute, each B_i scans its list L_i for *bad* entries $\{A,Q,V,t\}$ where ($CurrentTime$ - t) is too large to be acceptable. When a bad entry is found, the entry is removed from the list, and a bad access attempt event is triggered for user A.

Note that as an optimization, Alice need only compute and send a single signature to authenticate a list of all recent Q values to all servers.

4.3 Performance Improvement

There are several factors to consider when comparing the FK1 protocol to ours, including the cost of the group arithmetic for the basic blinding functions, the cost of related verification functions, and the cost, benefits, and risks of using a server-authenticated channel to each server.

Cost of Blinding Operations. With security factors $j = 80$ and $k = 1024$, the new protocol provides significantly higher performance than the FK1 protocol. Using $q = (p-1)/2$, each FK1 server must perform one 1023-bit exponentiation and the client must perform two.

When using $p = 2rq + 1$ as shown our method, we're using a subgroup of order $2^{160} > q > 2^{159}$. In the latter case, two client and one server computations are reduced to roughly 1/6 of the former amounts. (Note: Given the differing ways to evaluate equivalent symmetric and asymmetric security parameters, your mileage may vary.)

However, the client benefit is not realized when Alice must perform the 864-bit exponentiation in $g_P := h(P)^{2r}$. Yet, some of the benefit can be reclaimed in alternate constructions of g_P, such as the one described below.

This comparison so far ignores any added savings that can be achieved by eliminating the setup of the server-authenticated channel. However, by eliminating all server-authentication, the savings may come at the expense of allowing a little online guessing by false servers, and perhaps revealing the identity A to eavesdroppers.

Alternate Construction of g_P. We now suggest the alternate construction $g_P := g_1 \cdot g_2^{h(P) \bmod q}$. This uses fixed parameters g_1 and g_2 which are two random elements of order q with no known exponential relationship to each other. One possibility for creating universally acceptable values for g_1 and g_2 is to use a hash function as a random oracle, as in $g_1 := h(\text{"g1"})^{2r} \bmod p$. A similar technique is used to create acceptable parameters in DSA [9].

With the same security parameters, the alternate construction requires three 160-bit exponentiations for the client, and one for the server, which reduces the client cost by 53% and the server cost by 84%, when compared to FK1.

Cost of Authentication Function. Our method above requires a digital signature for the user to prove authenticity of her set of blinded passwords. Fortunately, client signature generation can be done once to create a message that encompasses one or more recently sent password requests for all servers, and server signature verification is fast when using RSA.

Furthermore, to eliminate the cost of a public key signing operation on the client, Alice might instead "sign" her set of blinded passwords with a keyed message authentication code, using a shared secret key ($K_i = h(K_m||i)$) that she enrolls with B_i. In this case she enrolls distinct keys and constructs distinct signatures for each server.

4.4 Arguments for Security

Although a full theoretical treatment is beyond the scope of this paper, we present a few simple arguments for the security of this method.

Each key share is a strong secret. The crucial data for each share is stored only in the secret y_i value on a hopefully secure credentials server, and it is released only in the exponent of a modified Diffie-Hellman exchange. This calculation is modulo a prime of the form $p = 2rq + 1$, which severely limits the information that an attacker can obtain about y_i. All that can be determined by a probing attack is whether y_i has factors in common with $2rq$. But since y_i is random and all factors other than 2 are huge, the probability is vanishingly small. Thus, as noted in [11], the only information that can be determined is the low bit of y_i.

Alice leaks zero information about P in her blinded **request** messages, since for any revealed value there's an equal probability that it was generated by any given P. (This is discussed further in Section 4.5.) And even the presence of

additional data, like the P^{xy} values, does not help the attacker determine P, since the y values are unrelated to P by any data known to an attacker.

The chance of information leakage from Alice in her **confirm** messages to an enemy in control of the channel is negligible, since she will abort before releasing any useful information if she receives any invalid reply from a server. Due to the combining hash function, if one share of the master key is incorrect, then with overwhelming probability the combined key will be incorrect. And if the master key is incorrect, then by the same reason the verifier hash value will be incorrect. So if they do match, Alice can be sure that her master key is correct.

In Section 4.1 we stated that the communications channel does not have to guarantee the integrity of the U_K and $proof_{PK_m}$ values sent by a server. To see why, consider an evil party that fabricates these values and sends them to Alice. At worst, this enemy can either validate a single guess for the password in each run, or perform a denial of service attack. If the client is designed to be no more tolerant of bad guesses than the server [2], then these attacks are roughly equivalent to the possible attacks in the secure channels model. In both models an enemy can make a limited small number of on-line guesses, in at least one direction, and can cause denial of service by modifying or deleting messages.

Both the $h(P)^{2r}$ function and the alternate construction in Section 4.3 guarantee an element of order q, and protect against the password-in-exponent and short-exponent problems noted in [19] and [11].

4.5 Short Exponents

An alternate approach to reducing computation is to use shorter exponents. For example, in a group with $p = 2q + 1$, with a 1023-bit q, one might use exponents in the range $[1, 2^{160} - 1]$. The use of short exponents in Diffie-Hellman was discussed in [25]. When using short exponents, the Pollard lambda method is the most efficient known way to compute a random exponent x of g^x for some known fixed base g. A lambda discrete log computation requires about $x^{1/2}$ operations. Yet there are no guarantees that a simpler solution will not be found.

Consider an aborted protocol, where the user simply reveals a series of blinded passwords, and no other information is available to an attacker. When using a full-size evenly distributed random exponent $x \in_R [1, o(G_q)]$, the P^x values reveal zero information about P.

But when using a short exponent $x \in_R [1, m]$, $m \ll q$, the security may require an added assumption of computational intractability, and it is desirable to remove unnecessary assumptions. Whether this assumption is significant is an open question.

So, with the (perhaps unwarranted) concern that short exponents introduce an unwanted assumption, our recommended approach to reducing computation

[2] Fortunately, people tend to have a low tolerance for login failures, and are likely to complain to systems administrators about recurring problems. However, the client must be designed to insure that at least the user is made aware of all failures.

is to use a subgroup of prime order significantly smaller than the modulus. This approach is also used in DSA.

These methods, when compared to FK1, can provide at least an equivalent level of security against discrete log attack with considerably less computation.

4.6 Flexible Server Location

In our model, we do not presume a pre-authenticated secure channel between the client and servers, and thus we do not require the user to validate the name of the server to maintain the security of the password. This frees the user to locate the server with a rich set of insecure mechanisms, such as those commonly used on the Internet. These methods include manually typed (or mis-typed) URLs, insecure DNS protocols, untrustworthy search engines, collections of links from unknown sources, all of which together provide a robust way to locate the correct server, but none of which guarantees against the chance of connecting to an imposter.

The crucial point is that, whether or not SSL is used, the worst threat posed in this model is one of denial of service – which is always present in the same form in the pre-authenticated server model. The new benefit of our model is that the password is never exposed to unconstrained attack, even when the client is connected to an imposter, by whatever means.

4.7 Trustworthy Clients

As in most related earlier work, we must fundamentally assume that the client software is trustworthy. Client software has control of the user input/output devices, and if malicious, could monitor the password during entry and send it to an enemy, or misuse it for evil purposes.

Note that a client may be deemed trustworthy for short-term transactions, but not trustworthy to handle long-term secrets. For example, even trustworthy local storage may be backed-up or inadvertently replicated to a less trustworthy domain. It may be sufficient that the system merely have the ability to enter trusted states for specific intervals, and perhaps even guarantee a trusted path between the keyboard and the secure application, while at the same time not be able to guarantee long-term security for persistent storage.

The trustworthy client requirement applies equally to our method, to FK1, and even to non-password systems where user-to-server authentication is mediated by a client machine or device. The evil misuse threat applies to many smart card systems, where client software presents transactions to be signed by the card.

We further note that web browsers may permit the client to use software applications that are loaded and run on-demand from servers to which the user connects. Such practice raises further important issues beyond the scope of this paper.

5 Applications

This protocol is useful for authenticating roaming users and retrieving private keys for use in network applications. It is especially applicable when the client has no capability for persistent storage of keys, or if one merely believes that a set of credentials servers is a safer long-term repository for keys than the disk on a poorly managed workstation.

Yet, the method can also enhance non-roaming systems. When client storage is present, and if it is deemed to offer at least some minimum level of protection, splitting shares of the user's master key among both local and secure remote storage may be desirable.

6 Conclusion

We've presented what appears to be the first description of a password-only multi-server roaming protocol. It retrieves sensitive user data from multiple related credentials servers, without exposing the password to off-line guessing unless all servers are compromised, and without relying on prior secure channels to provide this protection.

The method improves upon earlier methods in being able to perform these secure transactions with less computation, using either ordinary or elliptic curve groups, with simpler client configurations, and fewer requirements for proper user behavior.

The protocol is useful for authenticating roaming and non-roaming users and retrieving private keys for use in network applications – or more generally, wherever passwords are used in network computing.

The author thanks the anonymous reviewers for their helpful comments.

References

1. S. Bellovin and M. Merritt, Encrypted Key Exchange: Password-based Protocols Secure against Dictionary Attacks, Proc. IEEE Symposium on Research in Security and Privacy, May 1992.
2. V. Boyko, P. MacKenzie and S. Patel, Provably Secure Password Authenticated Key Exchange Using Diffie-Hellman, Advances in Cryptology - EUROCRYPT 2000, Lecture Notes in Computer Science, vol. 1807, Springer-Verlag, May 2000.
3. M. Bellare, D. Pointcheval and P. Rogaway, Authenticated Key Exchange Secure Against Dictionary Attack, Advances in Cryptology - EUROCRYPT 2000, Lecture Notes in Computer Science, vol. 1807, pp. 139-155, Springer-Verlag, May 2000.
4. M. K. Boyarsky, Public-Key Cryptography and Password Protocols: The Multi-User Case, Proc. 6th ACM Conference on Computer and Communications Security, November 1-4, 1999, Singapore.
5. D. Chaum, Security without Identification: Transaction Systems to Make Big Brother Obsolete, Communications of the ACM, 28 (1985), 1030-1044.
6. Cohen, F., 50 Ways to Attack Your World Wide Web System, Computer Security Institute Annual Conference, Washington, DC, October 1995.

7. T. Dierks and C. Allen, The TLS Protocol Version 1.0, IETF RFC 2246, http://www.ietf.org/rfc/rfc2246.txt, Internet Activities Board, January 1999.
8. E. Felton, D. Balfanz, D. Dean and D. Wallach, Web Spoofing: An Internet Con Game, 20th National Information Systems Security Conference, Oct. 7-10, 1997, Baltimore, Maryland, http://www.cs.princeton.edu/sip/pub/spoofing.html.
9. FIPS 186, Digital Signature Standard (DSS), NIST, 19 May 1994.
10. FIPS 180-1, Secure Hash Standard (SHA), NIST, 11 July 1994.
11. W. Ford and B. Kaliski, Server-Assisted Generation of a Strong Secret from a Password, Proc. 9^{th} International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE, June 14-16, 2000.
12. A. Frier, P. Karlton, and P. Kocher, The SSL 3.0 Protocol, Netscape Communications Corp., Nov 18, 1996.
13. L. Gong, T.M.A. Lomas, R.M. Needham, and J.H. Saltzer, Protecting Poorly Chosen Secrets from Guessing Attacks, IEEE Journal on Selected Areas in Communications, vol.11, no.5, June 1993, pp. 648-656.
14. L. Gong, Increasing Availability and Security of an Authentication Service, IEEE Journal on Selected Areas in Communications, vol. 11, no. 5, June 1993, pp. 657-662.
15. L. Gong, Optimal Authentication Protocols Resistant to Password Guessing Attacks, Proc. 8th IEEE Computer Security Foundations Workshop, Ireland, June 13, 1995, pp. 24-29.
16. S. Halevi and H. Krawczyk, Public-Key Cryptography and Password Protocols, Proc. Fifth ACM Conference on Computer and Communications Security, 1998.
17. IEEE Std 1363-2000, IEEE Standard Specifications for Public-Key Cryptography, IEEE, August 29, 2000, A.11.1, p. 131.
18. D. Jablon, Strong Password-Only Authenticated Key Exchange, ACM Computer Communications Review, October 1996, http://www.IntegritySciences.com/links.html#Jab96.
19. D. Jablon, Extended Password Protocols Immune to Dictionary Attack, Proc. 6^{th} Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Enterprise Security Workshop, IEEE, June 1997, http://www.IntegritySciences.com/links.html#Jab97.
20. C. Kaufman, R. Perlman, M. Speciner, Network Security: Private Communication in a Public World, Prentice-Hall, 1995, Chapter 8: Authentication of People, p. 205, 3rd paragraph.
21. S. Lucks, Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys, The Security Protocol Workshop '97, Ecole Normale Superieure, April 7-9, 1997.
22. P. MacKenzie and R. Swaminathan, Secure Network Authentication with Password Identification, submission to IEEE P1363 working group, http://grouper.ieee.org/groups/1363/, July 30, 1999.
23. R. Perlman and C. Kaufman, Secure Password-Based Protocol for Downloading a Private Key, Proc. 1999 Network and Distributed System Security Symposium, Internet Society, January 1999.
24. J. Tardo and K. Alagappan, SPX: Global Authentication Using Public Key Certificates, Proc. 1991 IEEE Computer Society Symposium on Security and Privacy, 1991, pp. 232-244.
25. P. C. van Oorschot, M. J. Wiener, On Diffie-Hellman Key Agreement with Short Exponents, Proceedings of Eurocrypt 96, Springer-Verlag, May 1996.
26. T. Wu, The Secure Remote Password Protocol, Proc. 1998 Network and Distributed System Security Symposium, Internet Society, January 1998, pp. 97-111.