Some Remarks on Protecting Weak Keys and Poorly-Chosen Secrets from Guessing Attacks

Gene Tsudik   Els Van Herreweghen*

Communications and Computer Science Department
IBM Zürich Research Laboratory
CH-8803 Rüschlikon, Switzerland
{gts,evh}@zurich.ibm.com

Abstract

Authentication and key distribution protocols that utilize weak secrets (such as passwords and PINs) are traditionally susceptible to guessing attacks whereby an adversary iterates through a relatively small key space and verifies the correct guess. Such attacks can be defeated by the use of public key encryption and careful protocol construction. In their recent work, Lomas et al. investigated this topic and developed a methodology for avoiding guessing attacks while incurring only moderate overhead. In this paper we discuss several issues concerning the proposed solution and suggest modifications that remove some of the constraints (such as synchronized time and state retention by the server) and result in simpler and more efficient protocols.

1 Introduction

At the 1989 Symposium on Operating Systems Principles (SOSP), Lomas et al. presented a paper: *"Reducing Risks from Poorly Chosen Keys"* [3]. This eye-opening paper addressed a number of problems associated with relying on weak human-selected secrets for authentication, and introduced a new class of hostile attacks called **verifiable-text**. Verifiable-text attacks form a superset of known-plaintext attacks. However, while vulnerabilities to known-plaintext attacks are fairly easy to detect and correct, verifiable-text attacks are, in general, much more subtle and more difficult to avoid. The authors presented some examples of cryptographic protocols resistant to verifiable-text attacks and suggested a protocol testing methodology for detecting vulnerabilities to these attacks.

For the purpose of demonstration, the authors developed a sample protocol; it is illustrated in figure 1. In this protocol (hereafter referred to as the *LGSN protocol*), A and B are two principals wishing to communicate securely (or simply to achieve mutual authentication). They share no previous secrets, however, each shares a secret with a trusted third-party S, i.e., an authentication and key distribution server.

1. $A \Longrightarrow$ S $[A, B, N_a^1, N_a^2, C_a, (T_a)^{P_a}]^{K_s}$
2. $S \Longrightarrow$ B $[A, B]$
3. $B \Longrightarrow$ S $[B, A, N_b^1, N_b^2, C_b, (T_b)^{P_b}]^{K_s}$
4. $S \Longrightarrow$ A $[N_a^1, N_a^2 \bigoplus K]^{P_a}$
5. $S \Longrightarrow$ B $[N_b^1, N_b^2 \bigoplus K]^{P_b}$

....

6. $A \Longrightarrow$ B $[R_a]^K$
7. $B \Longrightarrow$ A $[f1(R_a), R_b]^K$
8. $A \Longrightarrow$ B $[f2(R_b)]^K$

$\bigoplus$ represents the exclusive-OR (XOR) operation.

Figure 1: LGSN Protocol

A begins by generating three nonces[1]: N_a^1, N_a^2 and C_a. Similarly, B generates its own set of nonces: N_b^1, N_b^2 and C_b. T_a, T_b are the timestamps and P_a, P_b are the password-derived keys of A and B, respectively. K_s is the public key of the server S. Upon validating the requests in messages 1 and 3, the server S generates a new key K and distributes it individually to each of A and B. The remainder of the protocol (messages 6-8) is a simple two-way authentication using K; it is of little interest hereafter.

Following the protocol description, an informal analysis of the protocol steps is given. It is not outright claimed, but may be inferred that the demonstrated protocol is not susceptible to verifiable-text attacks.

*In Proceedings of IEEE Symposium on Reliable Distributed Systems, October 1993.

[1] A *nonce* is an unpredictable, used-only-once quantity [5].

2 Attacks on LGSN Protocol

Presented with no further requirements, the LGSN protocol is, in fact, vulnerable to verifiable-text attacks on both P_a and P_b. The attack can be mounted by either an outside intruder (i.e., someone with the ability to record and generate messages) or a "legitimate" insider (i.e., A or B).

Since there exists no perfect clock synchronization method, we can assume a maximum allowable system-wide clock skew Δ. Furthermore, we assume that the authentication server S is stateless, i.e., it does not record the last timestamp generated by every principal that uses the protocol. (In other words, no per protocol-run state information is kept.) All of this seems consistent with the paper and constitutes common practice in the design of such servers.

We now suppose that the adversary records an entire run of the protocol between A, B and S. Then, he proceeds to replay message 1 of the recorded protocol run within the allowable time skew interval, i.e., between time T_a and $T_a+\Delta$. The server S, not recognizing the replay for what it is, replies to A (in message 4) with:

4' $[N_a^1, N_a^2 \bigoplus K']^{P_a}$

The adversary has previously recorded:

4 $[N_a^1, N_a^2 \bigoplus K]^{P_a}$

If the adversary is an insider (i.e., B) he can stop here. Knowing both K and K', B can now iterate through the possible password space of P_a and for each password guess decrypt both messages, XOR their respective second halves and compare the result to $(K \bigoplus K')$.[2] A match indicates the correct guess of P_a.

An outsider attack is a bit more involved. In addition to replaying a pre-recorded message 1, the adversary also intercepts the communication from S to B in message 2 and, pretending to be B, replies with a pre-recorded message 3:

3 $[B, A, N_b^1, N_b^2, C_b, (T_b)^{P_b}]^{K_s}$

Once again, since T_b is still valid, the server replies to B with message 6:

5' $[N_b^1, N_b^2 \bigoplus K']^{P_b}$

This message is also intercepted by the adversary. Recall that the adversary has prevously recorded

5 $[N_b^1, N_b^2 \bigoplus K]^{P_b}$

At this point, the adversary has to iterate through all possible choices of both P_a and P_b (on the order of $|P_a| \times |P_b|$ trials). For every pair of guessed passwords, $(\hat{P_a}, \hat{P_b})$, the adversary decrypts messages 4, 4', 5 and 5'. He then proceeds to XOR the second halves of 4 with 4' and 5 with 5' in order to obtain a pair of *candidate* values of $(K \bigoplus K')$. If the two values match, both passwords are correctly guessed. This attack is possible because, even though the adversary does not know K, K' or $(K \bigoplus K')$, he can exploit the particulars of the message construction to extract enough redundancy for the attack to succeed.

Of course, both types of attack become infeasible if the server S implements a replay detection mechanism. It is, perhaps, due only to a minor oversight that the requirement for replay detection was not included in the subject paper. The authors appear to have been aware of this vulnerability[3]. Moreover, a follow-on paper to the Lomas et al. paper [4] explicitly states the requirement for replay detection by suggesting that the server store messages 1 and 3 for the duration of the maximum client-server clock skew. In fact, the motivation given for this replay detection measure is none other than the verifiable-text attack described above.

In the remainder of this paper, we discuss the importance of replay detection in key distribution protocols. Then, based on the LGSN protocol, we develop a simpler and more concise key distribution and authentication protocol that is resistant against verifiable text attacks while not requiring the server to implement replay detection.

3 Timeliness and Replay Detection

Detection of replayed or simply old messages is of utmost importance in authentication protocols. This is indisputable because authentication implies timeliness and an undetected replay may result in the protocol being completely defeated.

It is not clear that the same reasoning is applicable to key distribution. For example, an authentication protocol between two principals that do not share a key typically involves mediation by a mutually-trusted third party (the key server) with whom each principal shares a key. Such protocols, typified by the LGSN protocol in Figure 1, begin with a key distribution phase and end with direct mutual authentication between the two principals involved. While timeliness is an important factor to the two principals, it is not nearly as important to the server. This statement may appear *unorthodox* at first, however, the goal of a key distribution protocol is not to authenticate each of the two principals to the server. As long as the protocol provides assurances that the new key issued by the key server cannot be obtained by an unauthorized party, the server issuing the new key does not need to concern itself with the timeliness of the request.

Of course, there are other, more traditional motivations for implementing replay detection measures[4]. For example, it is considered, in general, good practice to log replayed messages in order to diagnose potentially suspicious behavior. On the other hand, replay detection does not necessarily have to take place in real time. It can even be considered a waste of server's time if its only purpose is the flagging of potentially suspicious activity. One viable alternative is for the server to simply log all (not just suspected replays) incoming messages for further processing. A different entity, e.g., an audit server, can be charged with scrutinizing event logs for signs of suspicious activity. (The audit server can even operate in parallel with the authentication server.)

Another important issue is the cost of replay detection. Even the most efficient methods of replay de-

[2] Since $(N_a^2 \bigoplus K' \bigoplus N_a^2 \bigoplus K) = K' \bigoplus K$.

[3] T. Mark A. Lomas, private communication, January 1993

[4] The term *replay detection* is meant to encompass the detection of old (but not necessarily replayed) messages

tection (e.g., [8, 9]) require keeping a certain amount of *soft* state in the server. This certainly makes the underlying protocol less robust (not to mention less elegant.)

There are also more subtle negative consequences of the use of timestamps for authentication. One notable peril of using timestamps is discussed by Gong in [7]. It involves a faulty clock setting on the part of the timestamp generator (which may be caused maliciously or accidentally) such that the clock is incorrectly set to some future time $T_f = (T_c + \Delta)$. An authentication message bearing a timestamp that corresponds to this clock reading will be undoubtedly rejected by the other party and the clock may be eventually reset back to normal. However, the erroneous message may have been recorded by an intruder who will patiently wait until the recorded message *ripens*, i.e., the current time becomes $T_c = T_f$. Then, the intruder simply replays the recorded message and obtains desired authentication. It is worth noting that this type of attack cannot be detected by most replay detection schemes (including the one suggested in [4].)

4 Some Issues Pertaining to LGSN Protocol

The main reason given in [4] for keeping state in the server is the protection against verifiable-text attacks of the type described in Section 1 above. While the proposed solution is clearly workable, it involves measures **outside** the protocol, i.e., state in the server is not represented in the actual protocol.

Without replay detection, the LGSN protocol is quite difficult to patch. One may be tempted to apply a quick fix by modifying message 4 in the following manner:

$S \Longrightarrow$ A $\qquad [N_a^1, (K)^{N_a^2}]^{P_a}$

However, the authors of [3] make an assumption that user-generated nonces can not be used as encryption keys. The reason for this assumption is not altogether clear. Perhaps the users (or workstations acting on their behalf) are not trusted to generate good keys. Yet, at the same time, they are trusted to generate reasonable nonces. The difference is subtle, at best.

It is not immediately obvious how the vulnerability in the LGSN protocol can be worked around or fixed. Thus, rather than try to fix protocol and run the risk of either introducing or discovering yet another subtle vulnerability, we chose to design *from scratch* a protocol which, while similar in spirit to LGSN, avoids time synchronization (and, therefore, state retention by the server) and addresses several other issues.

To summarize, the LGSN protocol raises the following concerns:

1. **Much public key encryption.** LGSN protocol requires six fields to be encrypted with the public key of the server. (A total of 12 fields taking into account key distribution to B.) Public key encryption costs are certainly not negligible. On other hand, many public key cryptosystems encrypt data in relatively large blocks (e.g., a reasonable block/key size for RSA[6] is 700 bits.) Therefore, the length of the message to be encrypted does not matter as long it is within the block size of the underlying cryptosystem.

2. **Key distribution is combined with authentication.** Message 4 (and 5) in LGSN includes some redundancy in the form of N_a^1 (N_b^1) which asserts to A (B) both data integrity and freshness of the entire message. This assurance is not strictly required since integrity and freshness of message 4 (5) are implicitly proven by the success of the following authentication between A and B using K.

3. **Many random numbers.** Both A and B have to generate three random numbers each. Random number generation is not typically considered as costly, however, many random number generators are encryption-based. Hence, each random number may cost one (or more) encryption operations. In this case, it is desirable to minimize the number of nonces and confounders needed in each run of the protocol.

4. **The protocol uses timestamps.** There are some well-known concerns (some of which were discussed in the previous section) that make timestamps an "unattractive" solution.

5. **Either the server must maintain state of recently-used timestamps or the protocol becomes vulnerable to verifiable-text attacks.** The LGSN protocol requires the server to keep state of recently-used timestamps. Keeping additional (per session) state can cost in terms of CPU cycles, storage, etc. It can also make the server less robust. It is well-understood, nevertheless, that logging and auditing of anomalous events is difficult (if not impossible) without keeping some state.

6. **Nonce-based version of LGSN needs two extra messages and requires the server to keep state.** In a nonce-based version of LGSN which is also presented in [3], A first requests a nonce from the server and subsequently includes this nonce in place of a timestamp (i.e., T_a in message 1 is replaced by the server's nonce.) Two additional messages do not represent a serious drawback; unless the server is very, very far away. The server must also keep state of its challenge (Ns) for the length of time between issuing the challenge and A's reply. If many requests come in *back-to-back*, the server would have to keep state of many challenges in volatile storage.

Finally, as discussed in the previous section, it is not necessarily important for message 1 to be *fresh*. Clearly, if the server is interested in logging old and replayed messages and raising alarms, freshness of message 1 is essential. Alternatively, if the server only cares about message 1 being generated by the legitimate party – A, then timestamps are no longer important. What remains important, however, is that:

1. No verifiable-text attacks on A's password-derived key (both outsider and insider attacks).

2. No known-plaintext attacks on the new key K (outsider only).

3. Impossibility of the adversary acting as the intended peer B.

5 LGSN Protocol Modified

Consider the protocol illustrated in figure 2. For the sake of brevity and conciseness, it includes only the interaction between A and S. The interaction between B and S (messages 2, 3 and 5) is substantially similar and the last stage of the protocol (messages 6-8) remains unchanged.

1.	$A \Longrightarrow S$	$A, B, \ [N_a \bigoplus B, N_a \bigoplus P_a]^{K_s}$
4.	$S \Longrightarrow A$	$N_a \bigoplus [N_a \bigoplus K]^{P_a}$

Figure 2: Modified LGSN Protocol

In this protocol, A begins by generating a single nonce, N_a. This nonce is used throughout the protocol as a *mask* that protects the user's password from verifiable-text attacks.

The goal of message 1 is to convey to the server that:

- **A generated N_a at some point in the past.** In other words, S must be able to extract N_a from the message and be sure that it was A that generated it. Consequently, A's secret, P_a must be included in the composition of message 1. However, it is not necessary for P_a to be used as an encryption key. Instead, it may simply be enclosed as part of the message but masked with the random value of N_a. However, only a single field carrying $N_a \bigoplus P_a$ is not enough to convince the server. N_a must appear twice in the composition of the message in order to provide enough redundancy for the server to authenticate its value.

- **A wanted to communicate to B at some point in the past.** This means that B's name must also be included in the encrypted portion of message 1. However, rather than simply including B's name, we chose to mask it with N_a in order to minimize the number of fields to be encrypted with public key K_s. Of course, this minimization is of only theoretical interest since many public key cryptosystems assume a relatively large key and data block sizes, e.g., 512-700 bits for RSA.

The addressing information is transmitted in the clear. It is not necessary to encrypt A's name as long as there is a one-to-one correspondence between names and passwords. Similarly, B's name appears in the clear, however, it is also included as in the encrypted portion. Leaving it out of the latter would open the protocol to attacks whereby an intruder could modify B's name with impunity.

The goal of message 4 is somewhat less ambitious. It must convey to A only that:

- **The newly-extracted key K is *fresh*, i.e., A has never used it before.**

- **No one but A and S can extract the same key.**

In order to assure *freshness*, the key is masked with N_a before encryption with P_a. This structure guarantees that the key is intertwined with the *fresh* nonce, N_a. If our concern was only with outsider attacks, then sending $[N_a \bigoplus K]^{P_a}$ in the clear would suffice (i.e., an intruder would need to know both N_a and P_a to extract K). However, in light of insider attacks, exposing $[N_a \bigoplus K]^{P_a}$ in the clear is dangerous. This is because the other party B knows the same key K. Therefore, it can guess A's password, decrypt the message, factor out a candidate value of N_a, and verify both N_a and P_a guesses by composing a version of message 1. For this reason, N_a is used, once again, as a mask that inhibits verifiable-text attacks and requires the intruder (both insiders and outsiders) to iterate on both N_a and P_a in order to verify respective guesses.

Note that, at this point, we are not concerned with whether message 4 has been created by S or whether the extracted key is the same as the one issued by S. These issues can be addressed by the subsequent two-way authentication between A and B using K.

6 Informal Discussion

The modified LGSN protocol cuts down on the number of encrypted fields. Public key encryption is confined to two data blocks as opposed to six in the original LGSN protocol.[5] Depending on the data block size of the underlying cryptosystem this may or may not result in any actual savings. Conventional encryption (e.g., DES[2]) is used only on a single block (in message 4) as opposed to three blocks in LGSN (one in message 1 and two in message 4). Furthermore, only one random number (nonce) needs to be generated by each party whereas LGSN requires three.

The server S is **stateless**, i.e., maintains no per-session information whatsoever. Furthermore, since timestamps are no longer used, no time synchronization and associated state-keeping is needed.

Any attack on P_a requires simultaneous discovery of N_a. An insider attack (i.e., B armed with the knowledge of K) would still require B to guess both N_a and P_a correctly. This translates into $|N_a| \times |P_a|$ trials. Assuming a reasonably strong random number generator, i.e., one that operates uniformly over a space of, say, 2^{36}, and a maximum weak secret space of 2^{20}, the protocol attains respectable strength of 2^{56} (which is roughly equal to the strength of DES).[6]

[5] The term *data block* refers to the size of a nonce, e.g., 64 bits. We also assume that this coincides with the block size of the underlying shared-key encryption function, e.g., DES[2].

[6] 2^{20} is the space of 6-digit PINs used for credit and cash cards in Europe.

The adversary can replay message 1 and obtain any number of replies of the type: $N_a \bigoplus [N_a \bigoplus K_i]^{P_a}$ (where K_i is the i-th session key generated by S.) Using this collected information, the adversary can obtain expressions of the type: $[N_a \bigoplus K_i]^{P_a} \bigoplus [N_a \bigoplus K_j]^{P_a}$ (for any i, j).

However, knowledge of such expressions is not useful even if the adversary knows both K_i and K_j (i.e., he is an insider.)

The protocol is not susceptible to verifiable-text attack on K. (At least, not without breaking Pa and Na at the same time.)

Other than discovering K from a legitimate protocol execution, the only way for an adversary X to impersonate B is by trying to change B's name in message 1 to his own, X. This would result in message 5 being encrypted under X's password P_x instead of P_b. This, however, is made impossible since any modification to B's name (outside of the encrypted portion) will be detected by the server.

One potential point of concern is that message 4 is not integrity-checked. In other words, A cannot be sure that the extracted key K is, in fact, issued by S. However, recall that a three-message two-way authentication protocol between A and B takes place immediately after the key distribution phase (see messages 6-8 in figure 1.) The "goodness" of message 4 is, thus, inferred directly from successful completion of $A <-> B$ authentication using K.

There may be circumstances where the integrity of message 4 is desired. For example, if the direct authentication between A and B does not immediately follow the key distribution phase (i.e., messages 1-5 are logically separated from messages 6-8), it may be important for both A and B to check the authenticity of K extracted from message 4 (or 5). The protocol can be augmented in many possible ways to support key integrity. For example, message 4 can be modified in any of the following ways:

- $N_a \bigoplus [N_a \bigoplus K]^{P_a}, \quad [K]^{K_s}$
- $N_a \bigoplus [N_a \bigoplus K]^{P_a}, \quad [N_a \bigoplus [N_a \bigoplus K]^{P_a}]^K$
- $N_a \bigoplus [N_a \bigoplus K]^{P_a}, \quad$ [contents of message 1]^K
- etc.

7 Variations on the Theme

7.1 Minimizing the Use of Public Key Encryption

The modified protocol discussed in the previous section reduces substantially the number of fields covered by public key encryption. Because of the relatively large input block sizes in many public key cryptosystems, this may not result in any practical benefits. However, assuming an arbitrary public key cryptosystem, it may still be of some value to minimize the amount of data to be encrypted.

Public key encryption is used to encrypt two data blocks in message 1. Obviously, not using public key encryption at all is not a viable choice since that would defeat our purpose of protection against verifiable-text attacks. Therefore, one of the remaining issues is whether a protocol using only a single-block public key encryption can be designed. One possibility is illustrated in figure 3. The present protocol appears to

1.	$A \Longrightarrow$ S	$A, B, \ [N_a \bigoplus P_a]^{K_s}, \ N_a \bigoplus [N_a \bigoplus B]^{P_a}$
4.	$S \Longrightarrow$ A	$N_a \bigoplus [N_a \bigoplus K]^{P_a}$

Figure 3: Protocol with One-Block Public Key Encryption

be resistant to verifiable-text attacks on P_a. However, unlike the protocol in figure 2 which requires on the order of $|N_a| \times |P_a|$ trials to break, this variation can be broken in $|N_a|$ trials (assuming that $|N_a| > |P_a|$).[7]

7.2 Can We Reduce the Use of Encryption Even Further?

So far, we have minimized the use of public key encryption. It remains an open issue whether the use of conventional encryption can be reduced further. We conjecture that it is impossible to construct a protocol with one-block public key encryption and one-block conventional encryption without *opening the door* for a verifiable-text attack. The intuitive reasoning behind this conjecture is as follows:

Suppose that there exists a key distribution protocol that has a one-block public key encryption in message 1 and a one-block conventional encryption in message 4. The encrypted field in message 1 must communicate (in secrecy) to the server an unpredictable, used-only-once value, N_a. It is this value that must be subsequently used along with P_a to distribute the key to A in message 4.

However, since message 1 carries only a single encrypted data block, the server **cannot** *be sure that N_a that it extracts from the message is genuine, i.e., generated by A. In other words, one-block encryption in message 1 lacks sufficient redundancy to authenticate the origin of N_a.*

Consequently, the adversary X can send to the server a bogus message 1 of the form

$$[\, A, X, \ [F(N_a, ...)]^{K_s} \,] \tag{1}$$

where F is an expression containing N_a and, perhaps, other variables. At this point the adversary knows $F(N_a, ...)$ but not necessarily N_a itself.

Since the server shares no secrets with A other than P_a and (presumably) N_a, the expression containing the new key in message 4 must have the general form $[G(N_a, K, ...)]$.

At least one of G or F must be based on P_a. Otherwise, the protocol can be trivially spoofed.

[7] After at most N_a trials, the adversary can obtain the value of $(N_a \bigoplus P_a)$ from a recorded message 1. Then, iterating on P_a, he can verify the correct choice by reconstructing $(N_a \bigoplus [N_a \bigoplus B]^{P_a})$.

Since the adversary X knows the value of K he can begin iterating through the password space and verify a correct guess by extracting a possible value of N_a from F, recomputing G and comparing it to $[G(N_a, K, ...)]$ received in message 4.

In order to demonstrate the subtle nature of this type of vulnerability we demonstrate (in figure 4) a sample protocol which uses only two encryptions. Details of the possible attack are left as an exercise to interested readers.

1.	$A \Longrightarrow$ S	$A, B,\ [N_a \bigoplus P_a \bigoplus B]^{K_s}$
4.	$S \Longrightarrow$ A	$N_a \bigoplus [N_a \bigoplus K]^{P_a}$

Figure 4: Protocol Susceptible to Verifiable-Text Attacks

7.3 Increasing Resistance to Verifiable-Text Attacks

Our proposed modifications to the LGSN protocol resulted in a more compact and concise protocol which does not require any state at the server while achieving the same resistance to verifiable-text attacks on the user's weak secret. However, as alluded to above, the strength of the protocol can be expressed as the cross-product of the weak key space and the strength of nonce-generating function.

A reasonable upper limit for a weak key space is around $10^6 \approx 2^{20}$ which can be thought of as a 6-digit PIN. The strength of the underlying nonce generator may vary. Therefore, it becomes desirable for the protocol to be flexible in order to accommodate variable-strength nonce generators. One simple and general approach is to introduce additional nonces as illustrated in figure 5. The example protocol with one additional nonce (N_a^1) requires on the order of $|N_a| \times |N_a^1| \times |P_a|$ trials to defeat. More nonces can be added in a similar fashion. Using this technique, the protocol can be extended to achieve any desired level of resistance to verifiable-text attacks.

1.	$A \Longrightarrow$ S	$A, B,\ [N_a^1, N_a \bigoplus B, N_a \bigoplus P_a]^{K_s}$
4.	$S \Longrightarrow$ A	$N_a^1 \bigoplus [N_a \bigoplus [N_a \bigoplus K]^{P_a}]^{P_a}$

Figure 5: Adding More Nonces

7.4 Reducing Password Use

One of the restrictions in [3] and [4] is that an end-device (such as a workstation) is not trusted to *generate* good keys. Consequently, workstation-generated nonces are not used as encryption keys. Only the server's public key K_s and the user's password-derived key P_a are used for encryption.

The reason for this restriction is not very convincing. If a workstation is trusted to generate good nonces it is not clear why it is not trusted to generate good keys. Of course, key generation in some public-key algorithms is a complicated and time-consuming procedure; e.g., key generation in RSA requires the computation of large prime numbers. However, in most shared-key cryptosystems, key generation is fairly simple; e.g., DES only adjusts bit parity and avoids a dozen or so *weak* keys (out the the total key-space of 2^{56}).

There is also the issue of the strength of the underlying random number generator (RNG). But, we already assume a reasonably strong RNG since the strength of all protocols discussed in this paper is based on the *goodness* of the workstation-generated nonces.

If we relax the above restriction, it is possible to reduce the use of the passwords in the protocol. In particular, we can avoid using P_a/P_b in messages 4 and 5.

In the protocol depicted in figure 6, an extra nonce N_a^1 is used instead of P_a to encrypt the key in message 4. The search space for an attacker trying to defeat the protocol is of the order of $|N_a^1| \times |N_a|$.

1.	$A \Longrightarrow$ S	$A, B,\ [N_a^1, N_a \bigoplus B, N_a \bigoplus P_a]^{K_s}$
4.	$S \Longrightarrow$ A	$N_a \bigoplus [N_a \bigoplus K]^{N_a^1}$

Figure 6: Encrypting The Distributed Key With A Nonce

One could get an impression that an additional merit of this protocol is in limiting password exposure. This is, alas, not so. The protocol simply minimizes the use of passwords. Breaking P_a is still linked to breaking N_a^1 and N_a: an attacker able to break message 4 needs only as few as $|P_a|$ extra iterations on message 1 to find P_a.

7.5 A Protocol with *Secret* Public Keys

One of the protocols described in [4] uses a novel approach to the environment where users are not expected to remember the server's public key. Instead, users are assigned individual public/private key-pairs and, upon request, each user is supplied with his very own public key encrypted with the password. The protocol is shown in figure 7.

One unusual aspect of this protocol is that the users' public keys are kept secret throughout. However, the protocol fails to take advantage of the very same feature in order to remove unnecessary redundancy. In particular, since each party's public key is distributed in secret, protected by that party's password, the mere possession of the appropriate public key can be used to authenticate the party in question. In other words, it is quite unnecessary to include P_a and P_b in messages 3 and 4. Combining this observation with our goal of minimizing state retention in

1. $A \Longrightarrow$ S A, B
2. $S \Longrightarrow$ A $A, B, N_s,\ [K_{sa}]^{P_a},\ [K_{sb}]^{P_b}$
3. $A \Longrightarrow$ B $[A, B, N_a^1, N_a^2, C_a, (N_s)^{P_a}]^{K_{sa}},$ $N_s,\ [K_{sb}]^{P_b}$
4. $B \Longrightarrow$ S $[B, A, N_b^1, N_b^2, C_b, (N_s)^{P_b}]^{K_{sb}},$ $[A, B, N_a^1, N_a^2, C_a, (N_s)^{P_a}]^{K_{sa}}$
5. $S \Longrightarrow$ B $[N_b^1, N_b^2 \bigoplus K]^{P_b},\ [N_a^1, N_a^2 \bigoplus K]^{P_a}$
6. $B \Longrightarrow$ A $[N_a^1, N_a^2 \bigoplus K]^{P_a}$

Figure 7: LGSN Protocol with secret public keys

the server, the protocol can be simplified as shown in figure 8.

1. $A \Longrightarrow$ S A, B
2. $S \Longrightarrow$ A $[K_{sa} \bigoplus B]^{P_a},\ [K_{sb} \bigoplus A]^{P_b}$
3. $A \Longrightarrow$ B $A, [N_a]^{K_{sa}},\ [K_{sb} \bigoplus A]^{P_b}$
4. $B \Longrightarrow$ S $A, B,\ [N_b]^{K_{sb}},\ [N_a]^{K_{sa}}$
5. $S \Longrightarrow$ B $N_b \bigoplus [N_b \bigoplus K]^{P_b},\ N_a \bigoplus [N_a \bigoplus K]^{P_a}$
6. $B \Longrightarrow$ A $N_a \bigoplus [N_a \bigoplus K]^{P_a}$

Figure 8: Simplified LGSN protocol with secret public keys

Acknowledgements

The authors are grateful to Li Gong, Mark Lomas and Roger Needham for the discussions that lead to the writing of this paper, and to Yves Deswarte, Phil Janson, Ralf Hauser and Liba Svobodova for their comments.

References

[1] W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, November 1976.

[2] National Bureau of Standards, "Federal Information Processing Standards," *National Bureau of Standards*, Publication 46, 1977.

[3] T. Lomas, L. Gong, J. Saltzer, R. Needham, "Reducing Risks from Poorly Chosen Keys," *Proceedings of ACM Symposium on Operating System Principles*, 1989.

[4] L. Gong, T. Lomas, R. Needham, J. Saltzer, "Protecting Poorly-Chosen Secrets from Guessing Attacks," *IEEE Journal on Selected Areas in Communications*, to appear in Spring 1993.

[5] R. Needham and M. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Communications of the ACM*, December 1978.

[6] R. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Communications of the ACM*, February 1978.

[7] L. Gong, "A Security Risk of Depending on Synchronized Clocks," *ACM Operating Systems Review*, Vol. 26, No. 1, January 1992.

[8] B. Liskov, L. Shrira and J. Wroclawski, "Efficient At-Most-Once Messages Based on Synchronized Clocks", **ACM Transactions on Computer Systems**, May 1991.

[9] G. Tsudik, "Access Control and Policy Enforcement in Internetworks", **Ph.D. Dissertation, USC Computer Science TR-91-15**, April 1991.