Technical Report

UCAM-CL-TR-445
ISSN 1476-2986

Number 445

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Secure sessions from weak secrets

Michael Roe, Bruce Christianson,
David Wheeler

July 1998

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

Secure Sessions from Weak Secrets

Michael Roe[1]
Bruce Christianson[2]
David Wheeler[3]

[1] Centre for Communications Systems Research, University of Cambridge
[2] Computer Science Department, University of Hertfordshire, Hatfield
[3] Computer Laboratory, University of Cambridge

Abstract. Sometimes two parties who share a weak secret k (such as a password) wish to share a strong secret s (such as a session key) without revealing information about k to a (possibly active) attacker. We assume that both parties can generate strong random numbers and forget secrets, and present three protocols for secure strong secret sharing, based on RSA, Diffie-Hellman, and El-Gamal. As well as being simpler and quicker than their predecessors, our protocols also have slightly stronger security properties: in particular, they make no cryptographic use of s and so impose no subtle restrictions upon the use which is made of s by other protocols.

1 Introduction

Sometimes there is a requirement to establish a secure session between two parties who initially share only a weak long-term secret. "Secure" includes the requirement that the parties can be sure that they are talking to each other, as well as properties of integrity and secrecy. By "weak secret" we mean a secret that is chosen from a moderately small set, so that an attacker could search through all possible values. Passwords are often weak secrets, as the total number of words in a dictionary is searchable.

A weak secret cannot be used directly as a cryptographic key to secure the session, as this is vulnerable to a known plaintext attack. If the attacker knows (or can guess with high probability of being right) the message plaintext m corresponding to a known encrypted text $E_k(m)$ then they can search through all possible values of the password until they find the one that yields the right value k. This reveals the password, which can then be used to decipher the session.

Suppose that the parties who wish to communicate have good random number generators. This means that they can generate secrets which are strong (chosen from a set which is too large to search) but not shared. We would like to have a protocol which starts with a weak shared secret and a pair of strong non-shared secrets and which ends up with a secret which is both strong and shared. We refer to such a protocol as a Strong Secret Sharing Password (S3P) Protocol.

Previous attempts at solving this problem include Bellovin and Merritt's Encrypted Key Exchange [2] and Lucks' Open Key Exchange [9]. A related, but slightly different approach is taken by Gong *et al* in [4]. In this paper, we present three new protocols for solving this problem, based on RSA, Diffie-Hellman and El Gamal respectively. As well as possessing slightly stronger security properties, our protocols have the advantage of being simpler and quicker than their predecessors.

2 Assumptions

In accord with tradition, we assume that the two parties trying to operate the S3P protocol are unambiguously known to each other as A and B.

We assume that neither party can reliably maintain the integrity of a strong secret s from one protocol run to another: in other words if A tries to use a strong secret from one run in another run, then there is a good chance that s either leaks, or is forgotten, or changes (or is changed) without A noticing that it has. This assumption may correspond to the fact that the parties move frequently from one piece of hardware to another, or because the hardware is initialized in some way between protocol runs to erase secret information. We discuss this issue further in the final section.

We assume that both parties can reliably maintain the integrity of public, slowly varying data such as software and public keys: other protocols are available to assist with this [8].

The protocols which we consider include the operations "generate a random bit pattern n" and "forget the bit pattern m". We assume that both ends have good irreproducible random bit generators and can forget secrets. By the first assumption we mean that our threat model does not consider the possibility of an attacker determining n by examining other bit patterns produced (previously or subsequently) by the same or other generators. By the second we mean that our threat model does not consider the possibility of an attacker subsequently determining m from from an examination of the hardware which has been instructed to forget it. Note that the hardware which must forget includes the random generator. This assumption is probably the most difficult requirement to realize in practice.

We turn now to describing the features which we desire the S3P protocol to have. At the start of the protocol A and B share a weak secret k. Following a run of the protocol which A believes to have ended correctly, it should be the case that B really did participate in that run of the protocol, and that the two of them do now share a fresh strong secret. The corresponding statement should also be true for a run which B believes to have ended correctly. The protocol should not reveal information about the weak secret k in any case.

The protocol should be secure against active attacks in which the attacker creates or modifies messages. Leakage or cryptographic compromise of a strong secret s shared using a protocol run should not reveal information about the password k. If several strong secrets s_i are shared by different runs using the same password k then obtaining one such s_i should not help an attacker to obtain s_j with $j \neq i$.

An attacker should not be able to obtain any information about whether a guessed value of the password is correct without making an active attack: effectively the attacker should be forced to masquerade as one of the participants to the other. An active attack should be detectable by at least one of the genuine participants, unless the guessed value is correct, and each such failed attack should eliminate no more than one possible value of the password (ie the unsuccessful guess) from the attacker's list of possible password values. Approaches such as [1] do not satisfy this requirement. Finally, if the password is compromised (by whatever means) this should not assist the attacker to obtain strong secrets agreed using the protocol with that password prior to the point of compromise, or to obtain subsequently agreed strong secrets by passive attack.

We wish to make no assumptions about what the strong shared secret will be used for. The S3P protocol run used to agree the strong shared secret s ends as soon as both parties can be sure that s has been appropriately shared. In the light of known chosen protocol attacks [7] we wish to impose no restrictions upon the nature of the cryptographic protocols or algorithms to which s is handed off for subsequent use, or upon the length of s itself. It may be intended to reveal s (for example s may be used as a one-time pad) or it may be that s is not intended to be used as a key at all, but as a salt or initial value. In particular, the S3P protocol should not assume that s is strong: it may be feasible for an attacker to search for s in the time available between steps of the protocol. This

may be because s is required to be weak (less than 40 bits, suppose) or because the protocol is running very slowly (the messages may be carried by a diskette sent through the post, for example.)

In the protocol descriptions that follow, we omit from each message the header information identifying which protocol is being used, which parties it purports to operate between, which run of the protocol the message relates to, and the sequence number of the message within that protocol run. Nor do we specify explicitly what conditions cause a participant to treat a particular run as having failed (e.g. receiving an incorrect bit pattern or timing out.) We shall consider these points further in section 6.

3 RSA based Protocol

A generates an RSA modulus $N = pq$ with p, q prime and so that $(p-1)/2, (q-1)/2$ each contains a large prime factor.

We assume that there is a publicly known function e which converts a password k into a large prime number $e(k)$ suitable for use as an RSA exponent. By large we mean that $e(k)$ always lies strictly between $\max p, q$ and $N - p - q + 1$. The function e could be implemented by some suitable algorithm (e.g. search for the next prime after $a + b.k$ where a, b are published constants guaranteed to exceed $\max p, q$.)

The protocol runs as follows:

$$A \rightarrow B : N \tag{1}$$

$$B \rightarrow A : z^{e(k)} \bmod N \tag{2}$$

$$A \rightarrow B : n_a \tag{3}$$

$$B \rightarrow A : n_b \tag{4}$$

Here $z = c|s|n_a|n_b$ where s is the session key, c is a strong random number called a *confounder*, and n_a, n_b are random numbers called *nonces*. The vertical bar | denotes concatenation, with the high order bits on the left. The presence of c prevents an attacker using a compromised session key and a copy of message (2) to search for k.

It is vital that there be no redundancy in the plaintext $z = c|s|n_a|n_b$ which is encrypted in message (2). If there were, then an attacker masquerading as A could use this to search for k in time to generate message (3) correctly and complete the protocol run. Note that consequently z must be a random number in the range $1 \ldots N$. The statistical distribution of the high order bits of c is thus skewed, because N is not a power of 2. However, the low order bits of c and all bits of s will not contain enough skew to be useful to an enemy after any achievable number of protocol runs. It must be infeasible for an attacker to search over the low-order bits of c. Otherwise after s is revealed for a completed run, passive search would reveal k by a match on message (2).

The purpose of messages (3) and (4) is to convince A and B that they are not experiencing an active attack. Instead of using the nonces n_a and n_b in messages

(3) and (4), we could use cryptographic hashes of them instead. But nothing is gained by doing this, and it requires us to exhibit a cryptographic algorithm with suitable subtle properties, a commitment which we prefer to avoid. A more dangerous alternative is to allow the contents of messages (3) and (4) to depend on s or c, for example by encrypting n_a or n_b under s. This is undesirable: cryptographic use of s in the S3P protocol may place subtle restrictions upon the cryptographic or other uses to which s may subsequently be put [7]. Similarly, we could derive s as a cryptographic hash of z, with similar objections. Worse still is to place encryptions with s of texts containing redundancy or known bit patterns in messages (3) and (4). This allows interactive breaking of the protocol between steps (3) and (4) by an attacker masquerading as B, who finds s in time to send message (4). Effectively, if s is not sufficiently strong then the attacker gets an undetected guess at k.

Factorization of N by an attacker gives the attacker k and, worse, allows the attacker to obtain old values of s. The public key N must therefore be many times longer than s, consequently a large number of bits is available for n_a, n_b and c.

Note that a fresh key N is required for each run of the protocol, but the function $e(k)$ can remain constant. Only A need verify the strength of the public key N. A must forget $d(k)$, the decryption key corresponding to $e(k)$, as well as p and q. Both A and B must forget c. B must forget the whole of z if the protocol fails at step (3).

In the RSA protocol, the key s and all nonces are chosen by B. Although s need not be strong, it must contain no redundancy and must not be predictable. Prior knowledge of the value of s which B will choose allows an attacker masquerading as A to determine k. Also note that s must appear random and so can't be a public key. Although the confounder c must be strong, the nonces n_a and n_b need not be strong, although they should be significantly harder to predict than k.

It is tempting to try and shorten the protocol run to three messages by combining the texts of messages (2) and (4) into a single message. This doesn't work, because there must be no redundancy in message (2).

We conclude this section with some remarks illustrating the constraints on the function $e(k)$. Suppose that e is a large fixed prime, and consider a broken variation of the RSA protocol where message (2) contains $z^e + k \bmod N$ in place of $z^{e(k)}$. An attacker masquerading as A chooses $N = pq$ where p is a prime of the form $r.e + 1$. Then the Euler totient $\phi(N) = re(q-1)$ so for almost all values of z we have $(z^e)^{r(q-1)} = 1 \pmod N$. Exhaustive search following a single foiled active attack now reveals k.

A similar argument applied to the original RSA protocol shows that if values of $e(k)$ contain prime factors much less than $\sqrt{N}$, then an attacker masquerading as A can eliminate several candidate values of k in a single run. Suppose p_i are small odd primes and the attacker would like to know which p_i divide $e(k)$. Define $p = 1 + 2\prod_{i \text{ even}} p_i, q = 1 + 2\prod_{i \text{ odd}} p_i$ and arrange the indexing of p_i so that p and q are prime. Setting $N = pq$ we have (almost certainly) that $p_i | e(k)$

iff $(z^{e(k)})^{P/p_i} = 1 \pmod N$ where $P = 4 \prod_{\text{all } i} p_i$.

To block attacks of this form it suffices to ensure that all $e(k)$ are primes with a one somewhere in the high order bits. This still allows an attacker to eliminate two values of k per active attack, but no more. We also need to ensure that the mapping from k to $e(k)$ is, as nearly as possible, one-to-one. Successive primes below N are almost always less than $\sqrt{N}$ integers apart.

4 Diffie-Hellman based Protocol

Let q be a publicly known large prime, and let g be a residue modulo q. To prevent various subtle attacks described in [2] and [6] we assume that $p = (q-1)/2$ is a prime and g is a generator modulo q, so that g^n has period $2p$. Note that in case $p \bmod 4 = 1$ we can take $g = 2$ [5, Theorem 95].

A and B select strong random numbers x, y respectively. By strong we mean that exhaustive search is infeasible. The protocol runs as follows:

$$A \rightarrow B : g^x + k \bmod q \tag{1}$$

$$B \rightarrow A : g^y \bmod q \mid n_b \tag{2}$$

$$A \rightarrow B : n_a \tag{3}$$

where we write $g^{2xy} \bmod q = z = c|s|n_a|n_b$ with semantics as in the RSA protocol.

Whereas the RSA-based protocol required four messages, the Diffie-Hellman variant can be done in three, effectively by combining both texts uttered by B in the same message. Consequently, in marked contrast to the RSA case, the second message in the DH protocol contains verifiable redundancy in the form of n_b. The reason an attacker cannot use this to break the protocol is that the redundancy is only detectable by an entity who knows x or y. These are not searchable by hypothesis, and the value of x is not deducible from message (1) even with a guessed value for k. Both A and B should check to ensure that $z \neq 1$. If $z = 1$ then the run fails, otherwise an active attacker could guess n_b and obtain s. The confounder c is required to prevent quadratic residue attacks from revealing bits of information about s.

As with RSA, eventual cracking of the chosen public key will give the attacker k and, worse, allow the attacker to obtain old values of s. For this reason the public key q must be many times longer than s, and so a large number of bits is again available for n_a, n_b and c. The nonces n_a, n_b and the confounder c need not be strong, although they should be significantly harder to predict than k. We require that x, y strong and large relative to $\log_2 q$.

A must forget x and g^x, while B must forget y. Both A and B must forget c. A must forget the whole of z if the protocol run fails at message (2). The DH protocol ensures that s appears random, but does not allow it to be chosen or predicted by A or B.

We conclude this section with some remarks concerning the choice of q, g. To avoid narrowing attacks, we require that q be a prime of the form $2p+1$ for

some prime p, and that g be a primitive root modulo q. Such keys are relatively expensive to generate, and in the DH protocol both A and B must check that q, g are suitable values, since using poor values can reveal k. In the RSA case only A need check. However, while the RSA protocol needs a new key N for each run, the DH protocol can use same key g, q many times. Consequently the key g, q could be relatively long-term and chosen by A and B jointly prior to the first run of the protocol, or else chosen, certified and published by some party trusted for this purpose by both A and B.

Alternatively, A could choose the public key and send it to B in the first message. B must carry out a deterministic test to verify that the key is of the correct form. A deterministic test should be used, since many non-deterministic tests assume random rather than malicious choice of candidate primes. To enable B to carry out such a test efficiently A can send a witness along with the key. However, we still need q to be many times longer than s. If A chooses q, g each run then it may be more efficient to find a prime q of the more general form $q = rp + 1$ where p is a large prime and r is relatively small, although it is then more difficult to find a generator g. The previous case corresponds to $r = 2$, whereas for this case $r = 2^n$ for a small n might be better. To avoid narrowing attacks when q is of this more general form, take $g^{rxy} \bmod q = z = h|s|n_a|n_b$ to force z into the large subgroup, and check $z \neq 1$. Another option is to replace $g^y \pmod q$ by $g^y + k \pmod q$ in message (2).

5 El Gamal based Protocol.

This variation of the DH protocol allows B to pick the session key and nonces, as was the case in the RSA protocol. The protocol runs as follows:

$$A \rightarrow B : g^x + k \bmod q \tag{1}$$

$$B \rightarrow A : g^y \bmod q \mid z \cdot g^{2xy} \bmod q \mid n_b \tag{2}$$

$$A \rightarrow B : n_a \tag{3}$$

where $z = h|s|n_a|n_b$ as for the DH protocol, except that we have some unused bits h instead of the confounder c.

This variant shares some features with the RSA case and some with the DH case. As with DH, it is a three-message protocol and the middle message must contain redundancy. However, while both the three and four message versions of DH work correctly, a four message version of EG formed by sending n_b in message (4) instead of in message (2) does not work: an attacker masquerading as B could choose y, w at random, send $g^y|w$ and then for each candidate value of g^x form a candidate w/g^{2xy} for z and check for a match on the third message to get k, s and n_b. It is interesting to note that the three message version of RSA does not work for similar reasons.

In the RSA protocol the value of s is chosen by B but s must contain no redundancy discernible to the attacker: otherwise k is in danger. The DH protocol

ensures that s appears random, but does not allow it to be chosen or predicted by the participants. The EG protocol allows B to choose s, and for s to contain redundancy in any form desired. Indeed for the EG protocol even prior knowledge of s by the attacker does not assist in an active attack against k. Also, in the EG protocol n_a and n_b may contain redundancy or known text. As with the other protocols, the nonces n_a and n_b need not be strong, although they should be significantly harder to predict than k. However in the EG protocol h really is not a confounder at all, and may contain any amount of known text or redundancy.

The EG protocol, like the DH protocol, can use the same key g, q many times. As in the DH protocol, both A and B must check that q, g are suitable values, since using poor values will reveal k. As with RSA and DH, eventual cracking of the chosen public key will give the attacker k and, worse, allow the attacker to obtain old values of s. For this reason the public key must be many times longer than s, and so a large number of bits is available for n_a, n_b, c.

A must forget x and g^x, while B must forget y. A must forget the calculated value of g^{2xy} and z if the protocol run fails at message (2). Apart from this, h need not be kept secret. The EG protocol allows s as well as n_a, n_b and even h to be chosen by B so as to contain redundancy or known text.

An alternative approach (which we do not recommend) is obtain n_a, n_b from g^{2xy} as in the DH protocol, rather than from z. This alternative requires z to contain redundancy, since otherwise an attacker can multiply z by some constant in the second message and not be detected, whereupon A and B do not share the same secret. But the form of redundancy to be imposed on z is problematic. For example, simple repetition of s would allow the attacker to rotate s left by one bit with a 50% chance of escaping detection. Similar problems arise with placing a known bit pattern in a part of h. Any redundancy involving a cryptographic algorithm (such as a hash function or encryption) may impose subtle restrictions on the protocols or algorithms which subsequently avail themselves of s. We avoid the need to solve this problem by incorporating the nonces into z. This solution imposes no redundancy upon s beyond what may be required for other reasons.

6 System Considerations

An important feature of all the S3P protocols we consider is that it is not acceptable to ignore an active attack. If active attacks are ignored, the attacker can make one active attack for each possible k, and will eventually succeed. If suitable emergency action is taken in the event of an active attack being detected (e.g. switching to a more expensive but physically secure channel, or to another, previously agreed, password, after a certain number of failed runs), then the attacker never gets enough information to improve his chances of guessing correctly by more than some previously agreed security parameter.

In an extreme case we can confine the attacker to two guesses, one with each of A and B. In a less extreme case, with (say) a million equally likely values for k, we could choose to allow the attacker 32 guesses with each of A and B. The attacker has less than a one in ten thousand chance of obtaining the true

value of k. Of course, this strategy requires some assumptions about the physical locations of A and B, and their ability to remember the number of active attacks detected over an appropriate time scale such as the expected life of the long term password k. We also need to specify, in any particular system context, how these numbers are stored and whether they are secret.

In particular, if the protocol is used by many pairs of participants, then an attacker can make a small number of attacks against each of a very large number of passwords, and will almost certainly succeed against one. The effects of this form of penetration, and the countermeasures for containing it, depend upon the interactions between the system-level protocols for which the strong shared secrets are used.

An attractive alternative to using a deterministic counter and a threshold is to invoke emergency action with a certain constant probability after each detected attack. For example, if we set this probability at 2% then the alarm will almost certainly be raised after 70 detected attacks, regardless of who detects them. Since we assume that all parties who use the S3P protocols are able to generate good random numbers, this stochastic technique imposes no new system constraints.

The primary system context which we consider for the S3P protocol is one of paranoia rather than hostility. In other words, we assume that the world is full of very clever and hardworking attackers, but at the same time we are confident that things will go right most of the time. In effect, we assume that the S3P protocol is nested inside another protocol which works nearly all the time except when there really is an active attack by an extraordinarily malicious and ingenious entity. The S3P protocol is intended both as a trip-wire to indicate whether the outer ramparts have been deliberately breached, and as a last-ditch defence.

One possible system context is where we wish to ensure that the right person is using a particular box. The box may be designed to be used by several different people (eg a workstation in a shared area) or by only one person (eg a hand-held device). The box may be stateful (eg able to retain session keys) or stateless (all mutable information is deliberately erased between uses). However a box may be stolen or tampered with. We wish to ensure that the box can only be used by a person who knows the correct password.

To deploy the S3P protocol we assume that the box is tamper-evident, and unviable to forge. We assume that the user checks the tamper-evident seal before entering the password at the start of each run. We assume that the box forgets the password once the S3P protocol run ends or fails, and that while the run is in progress the box is tamper-proof, in the weak sense that that the box will irrevocably destroy (forget) secrets rather than allow them to be read. This property might require that the box is used in a different environment from the one in which it is stored between runs.

Under these assumptions, the S3P protocol suffices to ensure that the box cannot be used by a person who does not know the password. In the case where the box may be used by more than one person, each person may have a separate

password. Note that tamper-proofing is not required except while the protocol is running. Tamper-evidence suffices the rest of the time even in the stateful case. State which persists between runs can therefore be used to support the outer "wrapping" protocol, so long as the state of the box between runs can reveal no information about the password.

The outer protocol must ensure that the inner S3P protocol is under no accidental illusions about whether an offered bit pattern represents an attempt to engage in the S3P protocol, and if so in which run, at what stage, and as whom. The S3P run must fail if any such presented bit pattern is incorrect, otherwise the enemy gets a free guess.

We also assume that "eventually" a run of the protocol which does not proceed will be regarded as having failed by at least one of the participants. However this timeout may be very long. One reason for this is that we do not wish to have too many false alarms, but there is another reason. We wish also to allow a system context in which a run of the S3P protocol is transported by a slow non-cryptographic outer protocol such as fax, snail-mail, or sneakernet. This gives rise to two further considerations: re-entrancy and interactive breaking.

If an S3P protocol run can take a long elapsed time, then the S3P protocol must be re-entrant. The total number of active runs (plus the number of previously detected failures) must be less than the threshold value for the number of active attacks which we are prepared to tolerate. This ensures that all runs which terminate successfully are safe. In particular, runs can be pipelined or used back-to-back between the same two parties on tamper-proof hardware such as smart cards which are kept locked up when not in use.

The possibility of a long elapsed time also provides one motivation for our consideration of the possibility that a value of s could be broken between steps of the S3P run, if it was used to encrypt a known plain text as part of the S3P protocol for example.

We conclude this section with a brief consideration of how to block a reflection attack. Suppose that A attempts to run the protocol with B. The attacker takes each message from A and replays it as if it came from B as part of a different protocol run. If A is not careful, she will end up sharing a fresh strong secret with herself, rather than with B, in violation of our requirement that the other intended participant must actually be involved in an apparently successful run. Of course, this attack cannot actually succeed against the protocols as we have described them here, since A always sends the odd-numbered messages and B the even. But suppose we wish to allow either party to initiate the protocol, possibly re-entrantly, so that A may legitimately use k to speak the lines attributed to B in the script.

We can block the reflection attack by associating the nonces firmly with principals. In respect of each password, one party is (by mutual agreement) the a–end and the other is the b–end. Suppose that *Carol* is the a–end and that *Ted* is the b–end. Then whenever *Carol* has to place a nonce in a message she always uses n_a, regardless of whether she is playing the part of A or of B.

7 Conclusion

Although the primary purpose of the S3P protocol is to share strong secrets, the design of the protocol does not assume that s is strong. The S3P protocol can also be used simply to allow a remote authentication service to authenticate a user to a "stateless" host which is local to the user. In this case s may be an authenticator for the audit trail. In our design we make no restrictions upon what the shared secret s is used for once the S3P protocol run has ended: s may be revealed, used as a one-time pad, a cryptographic key, as a salt or an initial value.

Our S3P protocols make no use of hash functions or symmetric cryptography. However our protocols rely completely for their properties upon the security of the public key systems used. Consequently it is necessary for the moduli to be uncrackable for at least the lifetime of all secrets (weak or strong) used or agreed with that modulus. This provides a sufficient number of bits to provide a strong secret and a strong confounder, together with two nonces. In contrast with the confounder, the nonces can be searchable, so long as the most likely nonce is less likely than some system threshold parameter.

Our protocols also rely upon the ability to generate random bit patterns, and to delete information irrecoverably. These are both interesting technical challenges. In particular the task of finding a suitable source of randomization, upon which (in the context of a particular system) it would be impractical to eavesdrop is one which would repay further study.

January 1998; revised July 1998.

Contact: Michael.Roe@ccsr.cam.ac.uk

References

1. Anderson, R., Lomas, M., 1994, Fortifying Key negotiation Schemes with Poorly Chosen Passwords, Electronics Letters, 30(13) 1040-1041.
2. Bellovin, S.M., Merritt, M., 1992, Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks, Proc IEEE Computer Society Symposium on Research in Security and Privacy, Oakland 92, 72-84.
3. Gong, L., 1995, Optimal Authentication Protocols Resistant to Password Guessing Attacks, Proc 8th IEEE Computer Security Foundations Workshop, 24-29.
4. Gong, L., Lomas, M., Needham, R., Salzer, J., 1993, Protecting Poorly Chosen Secrets from Guessing Attacks, IEEE Journal on Selected Areas in Communications, 11(5) 648-656.
5. Hardy, G.H., Wright, E.M., 1978, An Introduction to the Theory of Numbers, 5th edition, Oxford University Press
6. Jablon, D.P., 1996, Strong Password-Only Authenticated Key Exchange, Computer Communications Review, 26(5) 5-26.
7. Kelsey, J., Schneier, B., Wagner, D., 1998, Protocol Interactions and the Chosen Protocol Attack, Security Protocols 5, Springer LNCS 1361, 91-104.
8. Lomas, M., Christianson, B., 1995, To Whom am I Speaking? Remote Booting in a Hostile World, IEEE Computer, 28(1) 50-54.
9. Lucks, S., 1998, Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys, Security Protocols 5, Springer LNCS 1361, 79-90.
10. Steiner, M., Tsudik, G., Waidner, M., 1994, Refinement and Extension of Encrypted Key Exchange, Operating System Review, 29(3) 22-30.

This article was processed using the LaTeX macro package with LLNCS style