# Secure Sessions from Weak Secrets

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

[1] Computer Science Department, University of Hertfordshire, Hatfield
[2] Microsoft Research Limited, Cambridge
[3] Computer Laboratory, University of Cambridge
England, Europe

**Abstract.** Sometimes two parties who already share a weak secret $k$ such as a password wish to share also a strong secret $s$ such as a session key without revealing information about $k$ to an active attacker. We assume that both parties can generate strong random numbers and forget secrets, and present new 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 stronger security properties. In particular, our protocols make no cryptographic use of $s$ and so do not impose subtle restrictions upon the use which is subsequently made of $s$ by other protocols. Neither do we rely upon the existence of hash functions with serendipitous properties. In the course of presenting these protocols, we also consider how to frustrate some new types of cryptographic and system attack.

## 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 value $k$ which decrypts the cyphertext to $m$. 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 [3], Jablon's SPEKE [9], and Lucks' Open Key Exchange [12]. A related, but slightly different approach is taken by Gong *et al* in [7]. In this paper, we present new protocols for solving this problem, based on Rivest-Shamir-Adleman (RSA) [14], Diffie-Hellman (DH) [4] and El Gamal (EG) [5]. As well as possessing stronger security properties, our protocols have the advantage of being simpler and quicker than their predecessors.

In the next section, we give a careful specification of the properties which we desire S3P protocols to have, and of the assumptions under which we believe our protocols to have these properties. In Section 3, we present an S3P protocol based on RSA and show that it resists a number of attacks, some known and some novel. In section 4 we consider various DH-based S3P protocols which have stronger security properties than the conventional versions, and discuss some novel threats. In section 5 we consider the implications of using EG in place of DH when the requirement is to transfer an existing secret rather than to agree a new

one. In section 6, we discuss what should be done when an active attack is detected, and introduce the notion of a robust protocol wrapper. In the final section we summarize our conclusions.

## 2 S3P Protocol Properties and Assumptions

In this section, we give a careful specification of the properties which we desire S3P protocols to have, and of the assumptions under which we believe our protocols to have these properties.

The S3P protocols described in this paper are peer-to-peer protocols, which operate directly between two principals rather than between a principal and a server. 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 $B$.

### 2.1 Properties

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 really do now share a fresh strong secret $s$. 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.

### 2.2 Assumptions

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 may be because the hardware is initialized in some way between protocol runs to erase secret information. We discuss this issue further in section 6.

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 [11].

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 assumption 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 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 [10] 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.)

### 2.3 System Context

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. These data will be conveyed by the outer wrapping protocol, which will be discussed in section 6. Also, we have not yet specified explicitly what conditions cause a participant to treat a particular run as having failed (eg receiving an incorrect bit pattern or timing out.) We shall also consider these points further in section 6, but first we describe the inner S3P protocols themselves.

## 3 RSA based Protocol

In this section, we present an S3P protocol based on RSA. Correct protocols of this form have previously resisted construction, to the point where some have doubted their possibility. We show that this RSA-S3P protocol resists the known attacks as well as some novel ones.

### 3.1 Basic RSA-S3P Protocol Description

$A$ 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 the bit-lengths of $p, q$ and $N$ are prescribed exactly.

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 the bit length of $e(k)$ is strictly larger than that of both $p$ and $q$, and strictly smaller than that of $N$, for all candidate passwords $k$. The function $e$ could be implemented by some suitable algorithm. For example, supposing that the password $k$ is encoded in such a way that the bit length of $k$ is small relative to that of $\sqrt{N}$, then one suitable algorithm would be to search through ascending values of $i$ until a prime of the form $1 + k(a + ib)$ is found, where $a, b$ are published co-prime constants guaranteed to exceed $\max(p, q)$.

The RSA-S3P protocol runs as follows:

$$A \rightarrow B : N \tag{1}$$
$$B \rightarrow A : z^{e(k)} + 2^{e(k)} \bmod N \tag{2}$$
$$A \rightarrow B : n_a \tag{3}$$
$$B \rightarrow A : n_b \tag{4}$$

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 of bit strings, with the high order bits on the left.

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 $N$, although $B$ must check that $N$ has the correct bit-length. $A$ 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 $k$.

### 3.2 Design Discussion

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, this effect dies away exponentially with bit order, so the low order bits of $c$ plus all bits of $s$ cannot 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 [10]. 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 used in this way and is not sufficiently strong, then the attacker gets an undetected guess at $k$.

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).

### 3.3 Number-Theoretic Attacks

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 $c$. However the protocol must specify the exact bit-length of $N$, and $B$ must check that $N$ has the correct number of bits, in order to ensure that an attacker does not insert an extra factor into $N$ in order to gain residue information about $e(k)$.

The term $2^{e(k)}$ in the second message is required to block the *Bleichenbacher attack*: if this term is omitted, then an attacker masquerading as $B$ can send $z^e \bmod N$ in message (2), where $e = \prod_i e(k_i)$ and the $k_i$ are candidate passwords. Comparison of $n_a$ with $z^{e/e(k_i)}$ for each $i$ now reveals the correct $k_i$, and this can even be done in time for the false $B$ to generate message (4) correctly and finish the protocol run. Variations are possible in message (2), for example using $3^{e(k)}$ as the added term or using the exclusive-or $\oplus$ in place of modulo$-N$ addition.

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 RSA-S3P 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 $P = 4\prod_{\text{all } i} p_i$. Other number-theoretic attacks are considered by Patel [13].

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, since the bit-length of $N$ is prescribed by the protocol and checked by $B$. 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. The fact that $a$ and $b$ are relatively prime and $ab > N$ ensures this. The density of primes below $N$ implies that a prime will be found on average for $i$ of order $\ln N$ and almost always for $i$ very much less than $\sqrt{N}$. There are other subtle constraints upon the algorithm for $e$. For example, if the function $e(k)$ were instead defined to be the first prime after $a+b.k$, then the density of prime numbers implies that $e(k) = a + b.k + i$ for small $i$. The fact that $i$ is typically bounded by a small multiple of $\ln N$ gives the attacker information about $e(k) \bmod b$. An attacker pretending to be $A$ can choose $N$ to contain a factor of the form $br + 1$, and hence deduce information about $z \bmod br + 1$ which can be used to recover $z$ from message (2) with a greater probability of success than guessing $k$.

## 4 Diffie-Hellman based Protocol

In this section 4 we consider various DH-based S3P protocols which have stronger security properties than the conventional versions, and which make less use of superencypherment. We also discuss some novel threats.

### 4.1 Basic DH-S3P Protocol Description

Let $q$ be a publicly known large prime of prescribed length, and let $g$ be a publicly known residue modulo $q$. To prevent various known subtle attacks [3, 9, 13] 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 $g = 2$ [8, Theorem 95]. We assume that $k$ has a smaller bit-length than $q$ and is encoded in such a way that any two candidate values for $k$ differ in at least three bit positions. The last requirement is for technical reasons which will be discussed later, but can be guaranteed by adding a small amount of redundancy to $k$.

$A$ and $B$ select strong random numbers $x, y$ respectively. By strong we mean that exhaustive search is infeasible. The basic version of the DH-S3P protocol runs as follows:

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

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

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

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

Both $A$ and $B$ should check to ensure that $z \neq 0$ and $z \neq 1$. If $z = 0$ or $z = 1$ then the run fails, since otherwise an active attacker could masquerade as $B$ by using these values in message (2).

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 $B$.

### 4.2 Design Discussion

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 $k$.

The confounder $c$ is required to prevent quadratic residue attacks from revealing bits of information about $s$. Note that no superencypherment by $k$ is required in message (2).

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 parameter $q$ 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 purpose of the Hamming-distance restriction alluded to earlier upon the encoding of $k$ is to prevent the attacker testing multiple values of $k$ in a single run. If $g = 2$, then the attacker knows the discrete logarithms of small powers of 2, and can use this fact to test simultaneously a set of candidates for $k$, each of which differs in only one bit from some value $k_0$. The attacker sends $k_0$ as the first message and inspects $g^{2ix}$ for all $i$ less than the bit-length of $k$, to see if one contains $n_b$. If all candidate values for $k$ are at least three bits apart then this attack is defeated. For example including a Hamming code would add only 10 bits to a 1,000 bit $k$.

### 4.3 Choosing the Modulus

In this sub-section, we consider possible alternative approaches to the choice of $q$ and $g$. This in turn leads to some variations on the DH-based S3P protocols.

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 parameters 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 parameters $g$ and $q$ many times. Consequently the parameters $g$ and $q$ 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, protocol or algorithm trusted for this purpose by both $A$ and $B$.

Alternatively, $A$ could choose the public parameters $q, g$ and send them to $B$ in the first message. $B$ must carry out a deterministic test to verify that the parameters have the required properties. 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 parameters. 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, since these primes can be sieved for more quickly, 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 $g^{rxy} \bmod q = z = c|s|n_a|n_b$ to force $z$ into the large subgroup, and check $z \neq 1$ and $z \neq 0$. Another option is to replace $g^y \bmod q$ by $g^y + k \bmod q$ in message (2).

A further possibility is where a fresh $q$ of the prescribed length is chosen in some way for each particular run of the protocol. For example, the value of $q$ may depend in a deterministic way upon both $k$ and an unpredictable random value $r$, so that $q = q(k, r)$, where $r$ is produced during or just prior to the protocol run. This unpredictable value of $r$ need not be kept secret, and may published by a beacon, or agreed by $A$ and $B$ using some other protocol. In this case the protocol requires no superencypherment by $k$ at all. The first message contains just $g^x \bmod q$ with the provision that trial values for $x$ must be picked and discarded until one is found for which the high order bit of $g^x \bmod q$ is zero, and similarly for $y$. This ensures that the protocol run gives no information about which $q$ was used, and hence leaks no information about $k$. By forcing all $q(k, r)$ to have high-order bits $100\ldots0$ for some fixed number of zeros, we can make the probability of a high order one in $g^x \bmod q$ as small as desired.

### 4.4 Modified DH-S3P Protocol

We conclude this section by considering in more detail the case where the random number $r$ is produced during the S3P protocol run itself. The simplest method is for $A$ to send $r$ in the first message along with $g^x \bmod q$. However the protocol which follows is designed to illustrate a more paranoid scenario. We assume that $A$ and $B$ wish to use part of the value of $s$ to settle a bet [16]. Even if they have no doubt of one another's honesty, they must be able to prove to a sceptical third party that neither of them has the capability to influence the value of the shared secret $s$ in a predictable way. The random values such as $x, y$ actually generated during the course of the protocol run must be destroyed, and so cannot subsequently form part of an audit trail.

$A$ picks strong random numbers $m, x$ and $y'$. $B$ picks strong random numbers $m', x'$ and $y$. In the protocol description which follows, $q = q(0|m,k), q' = q(1|m',k)$ and $g, g'$ are the corresponding generators. We assume that $q = r.p+1, q' = r'.p'+1$ for large primes $p, p'$. The functions unzip0 and unzip1 denote the even and odd-numbered bits respectively.

The DHm-S3P protocol runs as follows:

$$A \rightarrow B : m \mid g^x \bmod q \tag{1}$$

$$B \rightarrow A : m' \mid \text{unzip0}\ ((g')^{x'} \bmod q' \mid g^y \bmod q) \tag{2}$$

$$A \rightarrow B : (g')^{y'} \bmod q' \tag{3}$$

$$B \rightarrow A : \text{unzip1}\ ((g')^{x'} \bmod q' \mid g^y \bmod q) \mid n_b \tag{4}$$

$$A \rightarrow B : n_a \tag{5}$$

Here $z = c|s|n_a|n_b = (g^{rxy} \bmod q) \oplus (g'^{r'x'y'} \bmod q')$ where $\oplus$ denotes bitwise XOR. Effectively the DHm-S3P protocol runs two instances of the basic protocol back to back, but reveals information only about the exclusive-or of the two results. This means that an attacker must crack discrete log for both $q$ and $q'$ simultaneously, rather than searching log tables one at a time. The unzip functions force $A$ to commit $y'$ before learning $g^{rxy}$, but after $B$ commits to $y$.

## 5 El Gamal based Protocol.

In this section we consider the implications of using EG in place of DH, when the requirement is to transfer an existing secret rather than to agree a new and unpredictable secret. The EG variation of the S3P protocol allows $B$ to pick the session key and nonces, as was the case in the RSA protocol. The EG-S3P 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 instead of the confounder $c$, $z$ contains a known fixed bit pattern $h$ chosen so that it is not invariant under shifts or subtraction from $q$. The constraints on $h$ will be further discussed below. The password $k$ is encoded as in the DH protocol. As in the Diffie-Hellman case, $A$ and $B$ should check that

$g^{2xy} \notin \{0, 1\}$. $A$ should also check that $z$ contains the expected value for $h$.

$A$ 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 $n_a, n_b$ to be chosen by $B$ so as to contain redundancy or known text.

The El Gamal 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. 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 $h$ is not a confounder at all. Instead, it contains redundancy to prevent a person in the middle modifying messages (2) and (3) in such a way that the protocol appears to complete successfully, but with $A$ and $B$ disagreeing on $s$. For example an attacker can multiply $z$ by two in the second message and shift $n_b$ and $n_a$ one bit to the left and right respectively, with a 50% chance of escaping detection. The result is that $s$ is shifted left one bit in transmission. Division and complementation are also possible. Such attacks can be prevented by placing a fixed bit pattern in $h$, for example a single 1-bit with $n$ 0-bits on either side will suffice, provided $2^{-n}$ is small relative to the chance of guessing $k$.

The EG protocol, like the DH protocol, can use the same parameters $g$ and $q$ 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 $n_a, n_b, h$.

An alternative approach (which we do not pursue here) is to obtain $n_a, n_b$ from $g^{2xy}$ as in the DH protocol, rather than from $z$.

## 6 System-level Considerations

In this section, we discuss what should be done when an active attack is detected, and introduce the notion of a robust protocol wrapper. We also discuss the system context for the deployment of S3P protocols, and the hardware support required.

### 6.1 Action following a Detected Attack

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 (eg 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.

### 6.2 Robust Protocol Wrappers

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 we call the wrapper, and that the outer wrapping protocol works nearly all the time unless there really is an active attack by an extraordinarily malicious and ingenious entity. The inner S3P protocol is intended both as a trip-wire to indicate whether the outer wrapper has been deliberately breached, and as a last-ditch defence.

A primary purpose of the outer protocol wrapper is to ensure that the inner S3P protocol is under no accidental misapprehension 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. A similar two-layer scheme for distinguishing accident from malice was used by Lomas and Christianson [11] and a related notion of robustness is discussed by Anderson and Needham [2]. 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, for example if it were used to encrypt a known plain text as part of the S3P protocol. In any protocol involving key agreement, it is possible to specify the agreement of a much longer

shared key than required, and to forget all but the required number of bits. Many applications of other published protocols would benefit from doing this.

Just as we do not assume $s$ to be strong, neither do we *require* $k$ to be weak: although the S3P protocols were originally designed to work with passwords $k$ drawn from a space of order $2^{20}$ possibilities, the protocols also have particularly nice properties when a 40-bit shared key $k$ is being traded up to a series of 120-bit keys $s_i$.

### 6.3 Tamper-proof Hardware Platform

One possible system context for S3P 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 mobile telephone or a hand-held authenticator). 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 user may have a different 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. However the inner protocol must not rely upon the outer protocol preserving state correctly. This point will be illustrated at the end of the section by discussion of a reflection attack.

### 6.4 Blocking Reflection Attacks

We conclude this section with a brief consideration of how to block a reflection attack. The inner S3P protocol is assumed to be stateless, and so a new run cannot reliably determine which other protocol runs are still active when it begins. Suppose that $A$ attempts to run the protocol with $B$. The attacker takes the first message from $A$ and replays it to $A$ as if it came from $B$ initiating a different run of the protocol. $A$'s reply to this is in turn reflected as a reply to $A$'s initial message, and so on for the subsequent messages. 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 any 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 Discussion

The protocols given in this paper are provocatively weak. For example, we use bit selection in place of a hash function, modulo addition to perform superencypherment, and a base of 2 for certain exponentiations.

This weakness is quite deliberate. From a practical point of view, our protocols could doubtless be strengthened by the judicious inclusion of "one-way" hash functions, or the use of more complex forms of superencypherment and convolution. We have instead put forward very concrete versions of the protocols, with primitives which rely upon specific number-theoretic relations between, for example, modular exponentiation, and addition or "unzip". Following Ockham, we wish to understand these simple protocols before we propose anything more complicated.

We have not provided correctness proofs for these protocols here. This is an area in which we anticipate future progress. The major present difficulty lies in determining precisely how the threat model interacts the

desired properties of the protocol with those of the underlying cryptoalgorithm. For example, one standard *reductio* approach might be to prove that, if the RSA-protocol reveals more information than it should about $k$, then it also gives an attacker the ability to decrypt unpredictable RSA cyphertexts. However attacks in the spirit of Bleichenbacher show that such an outcome need not constitute a break of the RSA cryptoalgorithm, and hence ought not simply to be presumed counterfactual. Conversely, a protocol continues to satisfy the assertion of a correctly proved predicate, even after the protocol has been broken by another means.

On the positive side, cryptographic innovations which we claim for this paper include the successful use of RSA as a vehicle for encrypted key exchange, and the modified versions of the DH protocols given in section 4.4. As a minor point, we also draw attention to the lack of superencypherment by $k$ in the second message of the basic version of our DH protocol. However, we regard as primary our original contribution to consideration of the system context given in section 6, including the introduction of robust protocol wrappers and their application to the case of "stateless" platforms.

The protocols in this paper owe an obvious debt to the original discussion by Bellovin and Merritt [3], which opened up a number of fertile research directions. We make no attempt to give a systematic account of all this related work here. (An excellent roadmap is provided by Jablon's website at `http://www.IntegritySciences.com`.)

Some of the material in this paper appeared in preliminary form in University of Cambridge Computer Laboratory Technical Report 458 (1998). We would like to thank Daniel Bleichenbacher, David Jablon, David Wagner and everyone else who provided attacks and related comments on these early versions of the protocols.

## 8 Conclusions

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. At a slightly more general level, we remark that it appears very difficult abstractly to model secu-

rity protocols in a formal way that takes adequate account both of the cryptographic properties assumed, and of the security service provided. Protocols may legitimately be used in ways not explicitly considered by their designers, and the safety of the resulting applications can depend in an unknown way upon the safety of obscure number-theoretic hostages which were abstracted away in the construction of the threat model.

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. Again, we remark at a general level the need to balance the bit-budget carefully when tuning the performance of security protocols which use public key cryptography.

Our S3P protocols also rely upon the ability of those using them to generate irreproducible 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. As a final remark of a general nature, we stress the importance of explicit consideration, when specifying the threat model, not only of the hardware platform supporting the security protocol, but also of the system context, and the security policy under which the hardware will be configured.

June 2000 Contact: B.Christianson@herts.ac.uk

## References


1. Anderson, R., Lomas, M., 1994, Fortifying Key negotiation Schemes with Poorly Chosen Passwords, Electronics Letters, 30(13) 1040–1041.
2. Anderson, R., Needham, R., 1998, Programming Satan's Computer, Springer LNCS 1000.
3. 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.
4. Diffie, W., Hellman, M., 1976, New Directions in Cryptography, IEEE Transactions on Information Theory, 22(6) 644–654.
5. ElGamal, T., 1985, A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, IEEE Transactions on Information Theory, 31(4) 469–472.
6. Gong, L., 1995, Optimal Authentication Protocols Resistant to Password Guessing Attacks, Proc 8th IEEE Computer Security Foundations Workshop, 24–29.
7. 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.
8. Hardy, G.H., Wright, E.M., 1978, An Introduction to the Theory of Numbers, 5th edition, Oxford University Press.
9. Jablon, D.P., 1996, Strong Password-Only Authenticated Key Exchange, Computer Communications Review, 26(5) 5–26.
10. Kelsey, J., Schneier, B., Wagner, D., 1998, Protocol Interactions and the Chosen Protocol Attack, Security Protocols 5, Springer LNCS 1361, 91–104.
11. Lomas, M., Christianson, B., 1995, To Whom am I Speaking? Remote Booting in a Hostile World, IEEE Computer, 28(1) 50–54.
12. Lucks, S., 1998, Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys, Security Protocols 5, Springer LNCS 1361, 79–90.
13. Patel, S., 1997, Number Theoretic Attacks on Secure Password Schemes, Proc IEEE Computer Society Symposium on Research in Security and Privacy, Oakland 97, 236–247.
14. Rivest, R., Shamir, A., Adleman, L., 1978, A Method for Obtaining Digital Signatures and Public Key Cryptosystems, Communications of the ACM, 21(2) 120–126.
15. Steiner, M., Tsudik, G., Waidner, M., 1994, Refinement and Extension of Encrypted Key Exchange, Operating Systems Review, 29(3) 22–30.
16. Wheeler, D., 1997, Transactions using Bets, Security Protocols 4, Springer LNCS 1189, 89–92.

# Secure Sessions from Weak Secrets (Transcript of Discussion)

Bruce Christianson

University of Hertfordshire

This story starts with a specific protocol, which we thought up in the Eagle back in 1997, and which was inspired by an earlier Security Protocol Workshop[1]. Here's the protocol, which is intended to leverage a weak shared password $k$ up into a fresh random strong shared secret $s$:

| | |
|---|---|
| $A \to B : (N, e)$ | an RSA public key |
| $B \to A : (z^e + k) \bmod N$ | where $z = c\|s\|n_a\|n_b$ is random |
| $A \to B : n_a$ | A knows $k, d$ |
| $B \to A : n_b$ | |

We thought it would be nice if we could *prove* that our protocol had the properties which we wanted to have. Then we discovered that there was a gap between what we could prove (or what other people did prove) and the kind of property that we actually wanted.

*The Scenario.* Here is the scenario in a little more detail. Alice and Bob have already agreed a weak secret (which I shall refer to as a password) and they want to agree a fresh strong secret. Neither of them can preserve both (1) the secrecy and (2) the integrity of a (3) strong bitstring in between protocol runs, but they both can meet any two out of these three requirements.

I won't go at any length into the motivation for this restriction. Maybe Alice or Bob is moving from one piece of hardware to another in between protocol runs. Maybe the hardware is unsecure between runs. Maybe Alice and Bob really are human. But a good application to have in mind for this kind of protocol is the

[1] Blake-Wilson and Menezes, Entity Authentication and Authenticated Key Transport Protocols, LNCS 1361, 137–158.

task of ensuring that the right person is using something like a mobile telephone. We look into the camera on telephones, we touch them with our fingertips, we type PIN numbers into them, we speak into them, it's a biometric paradise. Local authentication shouldn't be a problem.

However the best way to ensure that an attacker can't get an important secret out of the telephone when it's not being used is to ensure that there are no secrets in there in between runs. So Alice checks the hardware for tampering, types in her password and then runs the protocol with Bob (who might be a server) to agree a strong fresh key and perform mutual authentication. Then she can download all her bits.

This sort of scenario places further assumptions on the hardware: that the hardware can create good random numbers, do cryptography, exchange messages (subject to active attacks) and forget secrets. This last assumption is particularly problematic at the moment. I also assume that the implementations of the protocol are good, and in particular Alice and Bob perform all the specified checks, on the correct bit patterns.

Any assumption is really a statement that certain threats are going to be ignored. I'll come back to this point later. However subject to the assumptions listed there, the position paper contains several protocols that (we assert) do more or less what we want.

At the end of the run, Alice and Bob know that each of them has engaged in a protocol run with the other, and that they now share a fresh strong secret with each other and with nobody else.

**Question:** What are you assuming about the active attacker?

**Reply:** We assume that Eve can't access Alice or Bob's hardware while they're actually using it, and that Eve doesn't know the real password. Since the password is weak, the attacker could get lucky and guess it correctly. Then there's no hope. But we want to limit the guessing attack by forcing it online.

*Hash Functions with Mystic Properties.* We looked at what was done by people who analyse these kinds of protocols. The protocols which they analysed had

things like strong hash functions in them, which seemed to be there just in order to make the proofs work. Essentially, if you assume that the hash functions had certain mystic properties, then the protocol is correct. (Or equivalently, if the protocol doesn't have the required properties then the hash functions were not magic after all.)

But these papers didn't quite provide implementations for any genuinely magic hash functions. And we rather wanted to analyse the protocol that we actually had, which generates a particular set of bits at each stage. We can even tell you exactly what these bitpatterns are.

I'm one of those philistines who likes to understand a simple protocol before trying to understand a more complicated protocol. And as with any countermeasure, I want to know what value-added security the hash functions are actually buying. What threat are they protecting against. What attacks would succeed if they were not there?

Here's an example. The protocol I started with uses $n_a$ and $n_b$ as verifiers to prove knowledge of the shared secret $s$. Now versions which are formally analysed[2] usually use (in effect)

$$h_a(z), h_b(z), h_s(z) \quad \text{in place of} \quad n_a, n_b, s.$$

Here the $h_x$ are a family of hash functions with the necessary mystic properties, which we can deduce by looking at the proof.

There's no harm in the verifier space being searchable, so long as the verifier is much harder to guess than the password $k$. On the other hand, hash function source values mustn't be searchable.

[2] For an excellent example of an analysis of an RSA-based protocol similar to the one at the start of this talk, see MacKenzie, Patel and Swaminathan, Password-Authenticated Key-Exchange based on RSA, Proc Asiacrypt 2000, LNCS 1976 pp 599–613. For an analysis of a protocol based instead on Diffie-Hellman see MacKenzie, More Efficient Password-Authenticated Key-Exchange, Proc Top Crypt 2001, LNCS 2020 pp 361–377.

This gives the implementer a dilemma. If the implemetations of the hash functions really do have the properties required of the idealizations, then the more bits input to the hash the better.

But now suppose it turns out that my implementation of $h_x$ (and remember, this implemetation is not supplied with the kitset and so isn't covered by the warranty) is not really one-way. Then the protocol with the hash functions is less secure than the version without them. Indeed, the more things are added to the hash input, and the less searchable the verifiers are made, the more information the protocol leaks.

This sort of dilemma is a genuine practical problem. There are some valuable things to be learned from looking at practical details. For example, it's often a good idea to agree a key that's much longer than the one you actually need, and then throw most of it away, but you won't learn this if you just follow the logic, which usually doesn't address bit-budget issues carefully.

A reasonable compromise, given that $z$ is much longer than we need, would be to set $n_a = h_a(z_a)$ etc, where now $z = z_c|z_s|z_a|z_b$. But of course this in turn changes the nature of our reliance upon assumptions about the hash functions.

*Changing Assumptions.* Another reason for trying to keep protocols very simple is that a protocol implementation which works will often end up being used in a slightly different context. (laughter) It's so tempting isn't it.

For example I might decide that I don't want to assume that the fresh strong secret which the protocol shares is actually strong: I might want to use the S3P protocol to agree one or two bits that I'm going to hand off to another protocol to settle a bet[3]. I might not want to assume that the secret which I'm agreeing is going to be used as a secret key. It might be used in a way that makes it public as soon as it's agreed, for example as a shared pseudorandom seed. I might not want to assume that the password is weak. There was a time when I might have wanted to assume that the password was exactly forty bits long, and that I was using this protocol to agree a 128 bit key. I might want to change my

[3] D. Wheeler, Transactions using Bets, LNCS 1189, 89–92.

assumptions about what can be done in between runs with the hardware, and about interactive breaking attacks.

And as I said earlier, from the analysis point of view, each of my assumptions really corresponds to a constraint on the threat model. When I say, the system can create random numbers, what I'm saying is my threat model doesn't consider an attack where the attacker learns a later value of the random seed and uses that to work out what the pseudo-random sequence was yesterday.

When a proof assumes tamper-proof hardware that means that the proof doesn't consider any threat that involves tampering with the hardware. And when a proof assumes magic hash functions that means that the threat model just doesn't consider any cases where the magic wears off, potentially even years later.

Changing the threat model changes the properties which we need the hash functions to have. Inserting hash functions in such a way as to make a protocol have one set of properties, very often mucks up the other properties that the simple protocol might have had if the magic hash functions hadn't been added. I don't mind having to re-prove properties under new assumptions, or even having to restate the properties slightly. But I really don't want to have to re-implement the protocol. Think of the testing.

And at some point we really do have to implement these things. So how do we model the protocol, and how do we model all the threats, in such a way as to capture all the security properties that we might want it to have?

*Changing Abstractions.* The essential problem is that in real life we need to consider security protocols at several different levels of abstraction, and the various refinements which we need to make to get to an implementation typically do not respect the abstraction boundaries[4]

[4] For more on this theme see for example Chapter 10 of the book "Modelling and Analysis of Security Protocols" by Ryan, Schneider et al, Addison-Wesley 2001, ISBN 0201674718.

For example, the protocol I showed you at the beginning can be subjected to a narrowing attack. Suppose the part of Alice is played by Eve, and Eve deliberately chooses a bad value of $N$, which forces Bob into a narrow semigroup. For example, if Eve could choose $N = pq$ where $p = \alpha \cdot e + 1$ then $(z^e)^{r(q-1)} \bmod N = 1$, and so Eve can find the password by a process of elimination, possibly quickly enough to complete the protocol run.

One cure for this is to ensure that $e$ is relatively prime to $\phi(N)$, for example by requiring that $e$ be a prime with $e > N$, or alternatively[5] with $e^2 > N$ and $N \bmod e$ neither zero nor a divisor of $N$.

But it's really very hard to see how to model the protocol in a way that captures all the horrible potential for interactions between the number theory used by the cryptography and the protocol interleaving caused by the application semantics[6].

Another example is the shifting attack on the ElGamal version of the S3P protocol (this attack is described in section 5 of the position paper). Adding hash functions with appropriate properties would avoid the need to consider interaction attacks like this one explicitly. But understanding the properties which the hash function needs to have is enough to see how to block the attack. Then we don't need actually to put the hash functions into the implementation at all.

The dangerous temptation is to attempt to define away such, by inserting protocol elements such as random hash functions in such a way as to "remove" the possibility of bad interactions, and then to prove properties of clean abstractions instead of of the implementations which refine them[7].

[5] If prime $e$ divides $\phi(N)$ then either $e^2$ divides $N$ (impossible if $e^2 > N$) or else $e$ divides $p - 1$ for some prime factor $p$ of $N$. But then $p \bmod e = 1, N \bmod e = N/p \bmod e = N/p < e < p$. The condition is sufficient not necessary: 7 is relatively prime to $\phi(15) = 8$ but $15 \bmod 7 = 1$.

[6] For a splendid example in the symmetric key case see the presentation "Title?" by Anderson and Bond in these procedings.

[7] I should make it clear that the people who actually do the formal proofs are careful to a fault about making clear their limitations. The lovely papers by Phil Mackenzie

But there's another, possibly deeper, problem arising from the necessary use of abstractions.

*Protocol Layering.* When we deploy protocols like S3P, we typically want to layer them in with other protocols, or compose them with some sort of optimistic regime[8] or with some kind of fault-tolerance[9], or to hand off the artefacts (such as a fresh session key) to some other protocol which does something else with it. This introduces further subtleties into the modelling process.

I want to talk in a bit of detail about one important case, where the S3P protocol is used as a tripwire. In this case the S3P protocol is enclosed within an outer protocol wrapper. The outer protocol header on the front contains all the robustness fields that good sense says should be there: which protocol is this, which run does this message belong to, which message is this in that run, what's the hash of all the shared protocol states you've been in during this run, and so on.

Now there may be good reason to believe that this outer wrapper is itself a completely secure protocol, so there's no way that that Alice could not be talking to Bob to begin with. The inner key agreement protocol is being used as a tripwire to find out if anybody has breached the outer defences. It's both a tripwire and last line of defence: even if Eve can get into the Castle she still can't get into the Keep, and now Alice or Bob knows that she is inside, and maliciously so. If Eve trips over the tripwire then it's not an accident, it means she climbed over the dyke, swam the moat, scaled the battlements, and then tripped. Eve can't claim she was just on her way home. (laughter)

But this means that we need to have very clear models of what constitutes a failure. What constitutes somebody going over the tripwire? When does some-

---

which I mentioned in an earlier footnote, and the book "Modelling and Analysis of Security Protocols" by Ryan, Schneider et al, are models of good practice in this regard. The problems with maladapted refinement and abstraction boundaries tend to occur further along the food chain.

[8] LNCS 2845, 74–95.

[9] LNCS 2133, 155–169.

body intend to present a particular bit pattern to this protocol? What if a message got corrupted accidentally in transit?

Suppose Alice sends a message, it doesn't seem to get through, Alice sends it again, OK, now Bob receives the same message twice: is that a replay attack? Or just a protocol failure at a lower level? How do we model the interactions between the different levels of abstraction here? At what level(s) of abstraction do we log events?

What is a timeout, at the S3P level of abstraction? These messages may be travelling by fax, or by post, or by courier. There's a huge conceptual difference between a timeout of three seconds and one of three Megaseconds. It's not just a matter of putting a larger integer in a configuration file, it's a qualitatively different threat model. Similar remarks apply to interactive breaking and the level of re-entrancy, and I'll come back to this point in a minute.

*Non-deterministic Failure Detection.* What should we do when we detect a failure? One possibility is the Strangelove scenario: set off a doomsday device that will immediately bring the world to an end. That seems to be what usually happens in the model. But in real life, protocol failures are bound to happen eventually, probably more than once. Sometimes it's significant, and sometimes it's not. Somtimes we just have to push the car back up the cliff and see if it happens again.

For example, consider Eve's fundamental attack of attempting to guess the password $k$. It really isn't acceptable to allow no wrong guesses at all. Even users as experienced as Alice and Bob do occasionally mistype. But if you have a fixed limit (say three wrong guesses on a single user ID and you're out) then Eve can make just two guesses for each user. Provided there are enough users and their passwords are chosen semi-independently then Eve is sure to find somebody's password, and then she is in.

So there are advantages to taking non-deterministic actions on failure detection, because then the defenders don't need to collude. If everybody follows the

same probabilistic regime, then the alarm is raised with the acceptable degree of certainty after a distributed guessing attack above a certain threshold level.

Once again, the combination of layering of abstractions and non-determinism at some of the layers, means that it is not easy to see how to do the modelling with the current approaches.

Remember that non-determinism is a property of the model, the abstraction, not something which is "really" there in a fielded system. Consequently there's the potential for horrible interactions between the abstractions, just like the interactions between number theory and application semantics.

*Re-entrancy.* We've got to allow re-entrancy, because the S3P protocol might legitmately be run several times back to back. Alice establishes a one-way session with Bob, Bob immediately establishes a call-back channel with Alice, and so on. But maybe what appears to be Bob is actually Eve, using message reflection to get Alice to act as an oracle and tell Eve what the correct response is (see section 6.4 of the paper).

In this case, at the end of the run Alice shares the secret only with herself. True, nobody else knows the secret, but remember one of our primary security requirements was that if Alice thinks she has successfuly completed a protocol run with Bob then she must have done so. Now Alice is going to be willing to swear that Bob was alive at 9.30 yesterday evening, when in fact he was murdered that morning.

But we need the boxes running the protocol code to be stateless, or rather we don't want to assume that the state mechanism is reliable. We don't want to have to retain or correlate state between protocol runs, except for performance reasons.

There are various nice solutions, but which the usual models would not only obscure but actually make harder to use. You've got to model the state at different levels of abstraction, and you've got to model persistence of state at each level, and the interactions between the levels.

The idea is that protocols should proceed optimistically, to get good performance. Verification can usually afford to go more slowly[10] Usually everything will turn out to have worked OK, and when you find out that it didn't you have a possibly non-deterministic action[11] which you take when that happens.

In this way you can even tell whether the violation of the inner state is deliberate or accidental[12], a point which the conventional analysis doesn't catch.

*Every proposition is affirmative, therefore no proposition is negative.* To conclude, here is another version of the RSA protocol which I showed you at the begining of this talk.

$$
\begin{aligned}
&A \to B : (N, e) \\
&B \to A : (z^e + k) \bmod N, w \\
&A \to B : n_a, v \qquad\qquad \text{where } v^e = w \pmod N \\
&B \to A : n_b
\end{aligned}
$$

Usually in order to prove that a protocol like this is correct I would prove something like $P \Rightarrow Q$, *if* Eve can learn more about the password than the fact that her guess was wrong, *then* RSA is broken in the sense that Eve can decrypt unpredictable plaintext, or something like that. But for this protocol it is trivial to prove the proposition $Q$ in the form:

$$
\begin{aligned}
&\forall (N, e) \text{ chosen by Alice} \\
&\quad \forall w \text{ chosen by an independent referee and given to Eve} \\
&\quad\quad \text{Eve can determine } v : v^e = w \pmod N
\end{aligned}
$$

whence $P \Rightarrow Q$ for any $P$.

Far from establishing the correctness of this particular protocol, however, this 'proof' doesn't help at all to determine whether or not the protocol is broken.

---

[10] Again, see LNCS 2845 pp 74–95 for details.

[11] Remember once again, that non-determinism is a property of the abstraction, not the thing. See LNCS 1796, pp 60–64.

[12] Mark Lomas and Bruce Christianson, To Whom am I Speaking?, IEEE Computer 28(1) 1995, 50–54.

Is it? I wanted to to finish by having a dig at the random oracle model but I'll leave the details as a homework exercise.