Detecting failed attacks on human-interactive security protocols

A.W. Roscoe
Oxford University Department of Computer Science
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK
E-mail address: `Bill.Roscoe@cs.ox.ac.uk`

Abstract

One of the main challenges in pervasive computing is how we can establish secure communication over an untrusted high-bandwidth network without any initial knowledge or a Public Key Infrastructure. An approach studied by a number of researchers is building security though involving humans in a low-bandwidth "empirical" out-of-band channel where the transmitted information is authentic and cannot be faked or modified. A survey of such protocols can be found in [21]. Many protocols discussed there achieve the optimal amount of authentication for a given amount of human work. However it might still be attractive to attack them if a failed attack might be misdiagnosed as a communication failure and therefore remain undetected. In this paper we show how to transform protocols of this type to make such misdiagnosis essentially impossible.

1 Introduction

Human interactive security protocols (HISPs) achieve authentication by having one or more human users form part of a non-spoofable out-of-band (oob) channel between the devices that are involved. They do this without involving PKIs, shared secrets or trusted third parties (TTPs). The fact of humans' involvement severely limits the amount of data transferred on the oob channel, meaning that a compromise is required between the certainty of the authentication supplied by the protocol and ease of use. Therefore practical implementations frequently have a small, but not totally negligible, probability of a man-in-the-middle attack succeeding. For example if the humans have to compare six decimal digits, this probability cannot be below 10^{-6}. Protocols can be designed that essentially achieve this bound by preventing an attacker from performing combinatorial search to improve its chances.

A survey of the known approaches to creating such protocols can be found in [21]. They all work (in the sense of avoiding combinatorial attack) by the protocol participants becoming committed to one or more pieces of data such as a nonce, Diffie-Hellman token g^x or hash key before knowing that data. We give some examples as background (Section 2). In this paper we will only consider protocols that work in this way.

Though such protocols only give an attacker Eve a single guess against each protocol run between Alice and Bob, she will get further attempts if they try again after seeing that have a failed run. They are of course much more likely to try again if they believe the failure was due to a communication glitch, or a mistake on their part, rather than an abortive attack.

If it can be demonstrated to them that the strings they had to compare were different, then both they and the application which implements the protocol should know that an attack has occurred.

However if the attacker can abort before they have this information it may be impossible for them to distinguish the two possibilities.

In this paper we show that on published protocols the attacker can prevent certain knowledge of an attack, but that all the protocols we consider can be transformed so that any meaningful failed attack is definitely detectable.

In the next section we give background including a variety of protocols. In Section 3 we describe the mechanism behind our improvement, which depends on the addition of extra messages into the protocols based on the delayed decommitment of data from the protocol. Section 4 examines options for delayed decommitment, and potential attacks against them.

2 Background

This paper is set in the world of authentication protocols where either recipients need to authenticate the origins of messages, or where one or more people want to set up a secure network between a number of devices, but where there is no usable PKI or other pre-installed network of secrets to prove identities. Instead we assume that a high bandwidth network with no security guarantees at all is supplemented by out of band channels implemented by the human(s) using the devices. The assumption is made that these channels are authentic, in the sense that the humans know that they are correctly either transferring data between devices or correctly comparing pieces of data appearing on separate ones. We make no assumption, however that the out-of-band communications are private.

We assume that, in common with all the protocols in [21], the protocol succeeds (and authentication is assumed) if short strings generated on all the devices are equal in addition to all other messages conforming to the pattern expected of them. We will primarily concentrate on the case where this comparison is the final step of the protocol that confirms security. The reader will find other forms of protocol in [21].

It is easy to demonstrate that the strongest security guarantee that can be achieved with such a protocol is that an attacker can do no better than as a man in the middle who divides the group into two disjoint parts and runs the protocol separately with each, purporting to be the nodes missing for each (see Figure 1), succeeding with with probability $1/D$ where D is the number of short strings in the set from which the values compared via the humans are drawn. For the man in the middle will succeed in this strategy with at least this probability, on average, by using random values for the constants he has to introduce into the two partitioned runs, and without stronger guarantees on the network used to transport messages between the participants it is impossible to prevent such attempted attacks.

Protocols that meet this requirement are characterised by their use of techniques that prevent the man in the middle from choosing his constants more intelligently than by this random approach. For simplicity we primarily discuss the situation involving the pairing of Alice and Bob, but this discussion applies equally to situations like Figure 1, at least for protocols capable of handing groups.

The most basic and challenging aim of these protocols is to authenticate: either one party to other(s) or all parties to each other. Most of them take advantage of this authentication to agree a key between the parties. The intruder's aim is to have them believe they have completed the protocol with each other when in fact they have different values for the "agreed" data. He will succeed in this just when he gets Alice's and Bob's short strings to agree though they have not run the protocol properly with each other. The best protocols are designed so that he cannot perform a useful search to make them agree.

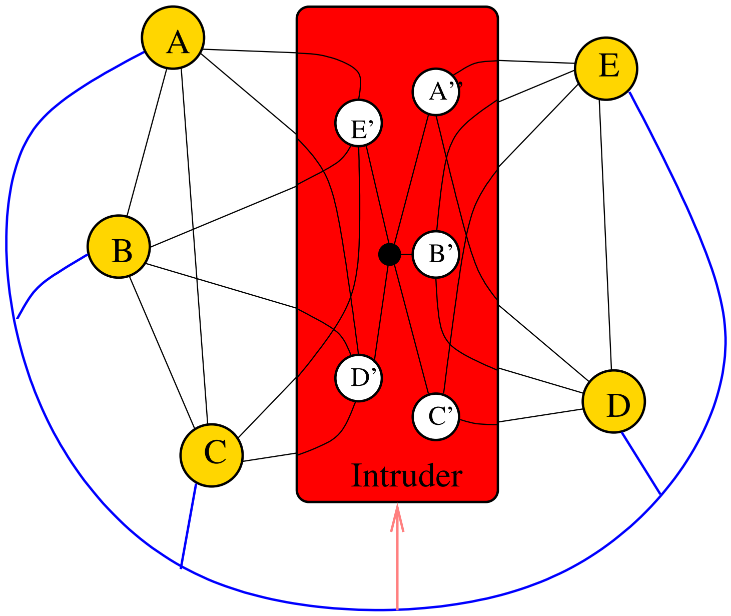

Figure 1: Attacker replacing intended group of 5 with two groups of 5 that have fake nodes.

More discussion of this issue, and examples of protocols susceptible to such attacks, can be found in [21]. The following are a few examples of protocols that are not susceptible in this way. We use a message-sending notation $A \longrightarrow_X B : M$ means that M is sent by A to B over the medium X. Two mediums are used: N is a high-bandwidth network with no security assumptions, typified by the internet. If A sends the message there is no guarantee that B gets it; if B apparently gets this message there is no guarantee that it comes from A (it might have been constructed by Eve); and even if the message gets through successfully it might have been overheard by Eve. In other words N is a *Dolev-Yao* channel [8]. E is the out-of-band, or empirical channel which is low bandwidth and implemented by human(s). It has the guarantee that if B thinks he has received M from A over E, then A really has sent this message to him.

The following protocol between A and B is adapted from the usual group version:

Symmetrised HCBK protocol (SHCBK), [18, 19]			
$1a.$	$A \longrightarrow_N B$	:	$A, INFO_A, hash(A, k_A)$
$1b.$	$B \longrightarrow_N A$	:	$B, INFO_B, hash(B, k_B)$
$2a.$	$A \longrightarrow_N B$	:	k_A
$2b.$	$B \longrightarrow_N A$	:	k_B
3.	$A \longleftrightarrow_E B$	:	$digest(k_A \oplus k_B, INFOS)$

Here, $\oplus$ is bitwise XOR and $INFOS = (INFO_A, INFO_B)$ is the information A and B are authenticating to each other. k_A and k_B are cryptographic length, e.g. 256, strings of random bits and *hash* is a standard cryptographic hash function. $digest(k, M)$ is a function that chooses a b-bit short string representing a short "hash" of M with respect to k. It must ideally satisfy:

Definition 1 [18, 19] *A b-bit digest function:* $digest : K \times M \rightarrow Y$ *where* K, M *and* $Y = \{0...(2^b - 1)\}$ *are the set of all keys, input messages and digest outputs, and moreover:*

- *for every* $m \in M$ *and* $y \in Y$, $Pr_{\{k \in K\}}[digest(k, m) = y] = 2^{-b}$
- *for every* $m, m' \in M$ $(m \neq m')$ *and* $\theta \in K$: $Pr_{\{k \in K\}}[digest(k, m) = digest(k \oplus \theta, m')] \leq 2^{-b}$

In practice the final 2^{-b} is often replaced by some ϵ slightly greater than 2^{-b}, and the θ is only required in some protocols, not including the pairwise version above. Digest functions are closely related to universal hash functions: some methods of computing them are discussed in [22].

This protocol works because each of A and B is completely committed to the value of its final digest before it releases its contribution to the digest key. They therefore each know that Eve had no knowledge whatsoever of the digest key in time to have sent them material that could have been the result of a search for a value that will give a particular digest result.

We can characterise the two keys k_A and k_B as *independent randomisers* of the short string, thanks to the properties of $\oplus$ and the specification of a digest. Choosing either value randomly, and independently of however the other is chosen, maps the short string to a random value in its domain. In this protocol each party chooses a randomiser r and ensures that it itself is committed to the value of the short string before it tells anyone the value of r.

[21] terms SHCBK a *direct binding* protocol because the agreed short string is a type of hash of the information Alice and Bob are trying to authenticate. This is analogous to the traditional use of signed cryptographic hashes in authenticating data: here the signature is replaced by the communication of the digest over the authentic oob channel, and the digest can be much shorter than a hash because of the care taken to avoid combinatorial attacks.

The following protocol, on the other hand, is termed *indirect binding* because the short string is chosen independently of the message and bound to it during the protocol. This particular protocol only transmits an authenticated message one way. Here, R_A and R_B are b-bit short strings chosen at random by the two agents.

Vaudenay pairwise one-way authentication protocol, [32]

$[A]$	$c \parallel d := commit(INFO_A, R_A)$,
1.	$A \longrightarrow_N B : INFO_A, c$
2.	$B \longrightarrow_N A : R_B$
3.	$A \longrightarrow_N B : d$
$[B]$	$R_A := open(INFO_A, c, d)$
4.	$A \longrightarrow_E B : R_A \oplus R_B$
	B verifies the correctness of $R_A \oplus R_B$

Here, $commit(M, R)$ is a pair $c \parallel d$ (following the notation of [32]) where knowledge of c commits the recipient to the values of M and R, through the said recipient may not yet know M and must not (even with the knowledge of M) be able to deduce the short string R. d reveals R and confirms (and if necessary reveals) M. Thus c is not a function of M and R, since if it were, a search through the relatively few possible values of R would reveal the real one. Rather it is in a one-to-many relationship with M and R, as possibly implemented by

- $c = hash(M, R, X)$ where X is a random nonce: a string of say 256 random bits.
- $d = (R, X)$

The purpose of R_B here is to make sure that if Alice sends Message 3 in response to a Message 2 faked by Eve before Bob has received Message 1, the chances of Eve's R_B equalling the real one are 2^{-b}. This normally prevents the final comparison being a success for Eve.

Note that R_A and R_B are independent randomisers for the short string, that B only sends R_B when it is committed to the final value of the short string (as well as $INFO_A$) and that A only reveals R_A when it is also committed to the final value of $R_A \oplus R_B$. Thus, though the appearance of this protocol is rather different from pairwise SHCBK, the *modus operandi* using a committed-before-reveal randomiser, and a pair of independent randomisers, is actually rather similar.

The two protocols above can be used to send or agree authenticated cryptographic material between Alice and Bob, but not in a way that is directly secret. Therefore the most obvious approach to building privacy for Alice and Bob is to make the authenticated material contain either an asymmetric "public" key or Diffie-Hellman style tokens such as g^{x_A} and g^{x_B}. Our final example is a direct binding protocol that uses cryptographic material in the authentication exchange: $g^{x_A x_B}$ becomes an authenticated session key between A and B. It is an interesting example chiefly because it is widely implemented [3]. The core of this protocol is:

ZRTP protocol (Zimmerman [33])			
1.	$A \longrightarrow_N B$	:	$hash(g^{x_A})$
2.	$B \longrightarrow_N A$	:	g^{X_B}
3.	$A \longrightarrow_N B$	:	g^{X_A}
4.	$A \longleftrightarrow_E B$	:	$shorthash(g^{x_A}, g^{x_B})$

(The shorthash can equally be of the shared key $g^{x_A x_B}$.) This works in a way very similar to the shortened pairwise version of SHCBK mentioned above. The only difference is that the randomisation present in the authentication re-uses the Diffie-Hellman exchange that establishes the key. This is clearly efficient, but the properties of modular exponentiation and the b-bit *shorthash* need to be carefully managed and studied to provably achieve the same security bounds attained by the first two protocols. In other words, X_A and X_B are both used as randomisers for this protocol, and while intuitively this should work well it is not mathematically obvious, as it was with the previous protocols.

In each of these protocols, it is the scheme of commitment to the randomisers that avoids combinatorial attacks based on searching, and they all, seemingly inevitably, use a method of *commitment before knowledge.* [Of course the protocols also have to be immune to conventional attacks possible without such searching.] One assumption implicit in these protocols is that all the cryptographically important values and operations used in them, other than short values for human comparison, are strong enough to withstand any imaginable brute-force attack with probability so close to one that we can disregard the difference.

3 Attack detection

Although these protocols prevent the attacker from searching against the short string on a single run of the protocol, they do not prevent repeated attempts to have a correct guess against multiple independent runs, whether the users try over and over again to bootstrap security in a single group of devices, or attempts are made against a variety of groups as they form until one succeeds. Realising this, we should think about how the attacker might execute a single attack. Again restricting for simplicity to the pairing of Alice and Bob:

1. When a run between Alice and Bob begins, initiated by Alice, the intruder Eve will play the part of Bob in that run and herself initiate a run with Bob, pretending to be Alice. These two runs will progress through a number of stages of interaction leading to the final comparison of short strings (which unbeknownst to them are the strings generated by the two separate runs) by Alice and Bob. Because we are assuming our protocol is optimal in the sense discussed above, Eve just introduces a random value when she has to contribute something like a key or nonce to either of her versions of the protocol. Of course the attack only succeeds if the separate short stings s_A and s_B in the two runs happen to co-incide.

2. At some point in the run between Alice and Eve (qua Bob), each of the two participants will have the data to allow them to compute their short string s_A. One of them will necessarily have this information before the other, inevitably the one to send the last message before they *both* know it. The same thing happens in the run between Eve (qua Alice) and Bob.

3. In organising her attack, Eve is free to interleave the messages she sends/receives to Alice and Bob so that their own order of sends and receipts exactly follows what would have happened if they had really been running the protocol with each other. If she does this, Eve is bound to have first knowledge of the digest in one of the two runs and second knowledge in the other. Furthermore Eve will know *both* s_A and s_B before one of Alice and Bob knows his or her own short string. In some protocols (particularly the more symmetric group protocols) it may be possible for Eve to choose a schedule so that she knows both of these values before any of the honest participants do. In pairing protocols she will be playing the two roles in the two runs: one of these will be the one that gets knowledge of s first.

4. It follows that Eve will know whether her attack will succeed or fail before all of the devices that are being have the information that will allow them to know. If she is going to succeed, she will naturally press on and break into what Alice and Bob will both regard as a successfully completed session. However if she is going to fail, the logical move for her will be not to send the messages remaining to Alice and/or Bob that will reveal the short string to them.

5. In that last case, Alice and Bob will never *both* have their short strings, and so will not be in a position to compare them. They will therefore lack conclusive evidence that an attack was taking place and may very well conclude that the failure of the protocol to complete is down to a communications glitch.

What would it take for Alice and Bob to be able to know that someone was attacking them? If it can be shown that some parties had progressed a long way in interacting with the two of them and then aborted runs which would have had differing short strings, this would represent conclusive evidence. Alice and Bob are still connected by their out-of-band channel, and so can compare notes on what has happened. Let us suppose we could achieve, following an aborted run, a state where each of Alice and Bob knows one of the following for certain:

- No conceivable intruder could have known the digest their own run was heading to at the point of the abort.
- Has direct evidence, through inconsistency of messages received, that an attack was attempted.
- Knows the digest value that he or she would have calculated and compared, and whether their device would have raised any further inconsistencies before the and of the protocol.

Then by conferring over the oob channel they can conclude one of the following *trichotomy* of outcomes.

(a) No intruder knew both short strings at the point of the abort, and therefore had no way of knowing at that point that any attack being carried out would succeed or not. An intruder would have no benefit from this other than denial of service.

(b) That the protocol failed even though the short strings would have agreed: this is almost certainly due to a communications failure.

(c) That Alice and Bob were heading towards different short strings, meaning that an attacker was almost certainly involved, or that some other sort of attack was taking place.

One might imagine that in the last eventuality Alice and Bob would take countermeasures such as

1. Upping the security level of any subsequent connection attempt (for example by increasing the length of the short string)

2. Notifying any servers associated with the applications running the protocol, so that they can take appropriate defensive or forensic action.

3. Changing the communications network being used.

We term a protocol that achieves this *auditable* because when things go wrong we can scrutinise the run and determine what when wrong.

The above trichotomy is unachievable in any of our example protocols thanks to our earlier analysis which showed that in the man in the middle attack the attacker can know both s_A and s_B before the moment when these values are known by Alice and Bob respectively, and can stop at least one of them knowing its value if they are not equal. What we will now demonstrate is a way of modifying the protocols so that they do achieve it: we introduce time-dependent data and time-outs into modified protocols. We can ensure that the new protocol behaves like the old one within some time limit T, and unless it completes before then either any intruder had no information about success before T or there is some later time T' at which Alice and Bob both know the short strings from their runs. There is no need to know exactly what the times T and T' are, merely that they exist. The best way to describe this is by modifying the three protocols set out above so that they have this property. We assume that there is an additional cryptographic primitive $delay(x, t)$ with the properties

- Anyone who knows $delay(x, t)$ can know x, but not until at least t units of time since its first creation.

- No-one can deduce anything about x from it before that point.

- In the same agent that created $delay(x, T)$, the boolean $intime(x)$ returns *true* if before T since the call of $delay(x, T)$, and *false* otherwise.

We will discuss potential implementations of $delay(x, t)$ in Section 4.

In the adapted protocol descriptions below the statement $[C]\, x := delay(k, T)$ is the creation of one of these time locked means of calculating x by agent C. The strategy we follow is to make sure that Alice and Bob both have in their possession the means to calculate their short strings *perhaps at a later time* before Eve knows both s_A and s_B.

We suggest four strategies below of how to use $delay(x, t)$ in modified protocols.

1. Observe that $delay(x, T)$ has the interesting property of both committing the recipient to the value x before any but the sender knows it, and later releasing it. Therefore there is no *a priori* need to send the commitment and opening message openly any more. Of course such a protocol cannot complete before all the delayed data has opened.

   The simplest protocol which works in this context is

Delay commit protocol

$[A]$	$d_A := delay(k_A, T)$		
1.	$A \longrightarrow_N B$	:	$A, INFO_A, d_A, hash(A, k_A)$
2.	$B \longrightarrow_N A$	:	$B, INFO_B, k_B$
$[A]$	$intime(k_A)$		
3.	$A \longleftrightarrow_E B$	:	$digest(k_A \oplus k_B, INFOS)$

Here A knows that she was committed to the final digest before the *delay* opened, meaning that the value received apparently from B cannot have aimed at creating any particular value of the digest. Similarly B knows that he is completely committed to the final value of the digest at the point he sends Message 2, and that he has randomised it. Therefore, although the attacker can send a fake Message 1 to B at any time (even after the *delay* has opened), there is nothing it can do to bias B's digest value.

Furthermore, if an attack is attempted where both digests are known to the attacker, then if A has not already abandoned the protocol on the time-out A also knows (i) her own value of the digest (because she will have received Message 2) and (ii) that if there was no attack then B will also know his value of the digest without further communication once T has passed. This allows auditing.

2. Replace the existing commitment mechanisms (i.e. hashes and *commit*) by instances of *delay*. Time out the protocol unless all delayed data is released in the open by the protocol before the delayed version of it opens. These extra messages can have advantages in protocol design, but because of the necessity of checking the consistency of the commitment against the second communications the users will still have to wait for the *delay*s to open. This latter inconvenience (and a perhaps a lot of work) will be avoidable if a receiving party can check for this consistency without having to open the *delay*. If, for example, $delay(x, T)$ is created using a function that is much easier to compute directly than invert, it will be easier for the receiving party, upon receiving x directly, to re-create $delay(x, T)$ and test for equality.[1]

   Depending on the means of opening a $delay(x, t)$, this (needed for both this approach and (1) above) might well be a lot of work. Generally we prefer to use $delay(x, t)$ in such a way that it does not have to be checked in a completed run of a protocol, only when auditing a failed run.

3. Accompanying each instance of a hash or *commit* in the original protocol with a corresponding *delay* (i.e. what was sent in the original protocol together with what was sent in the first approach above). This would achieve our aims completely provided suitable time-outs were included in the amended protocol, as above.

   This approach achieves our objectives, but we term it *Weak* because its security depends on the correctness of the *delay* operator. We will discuss in the next section why it may be wise *not* to rely on *delay* so much. Therefore we prefer the following approach.

4. Develop a protocol which is auditable in the sense that we achieve the trichotomy set out above, but does not depend on the correctness of *delay* (in other words it would still be secure, but not auditable, if the intruder were able to extract x from $delay(x, T)$ immediately.)

The adapted protocols we present below are designed in line with this final option.

[1]If the value x has had to be salted to make the *delay* secure, it would then be necessary for the direct communication of x to include the salt as well.

Auditable SHCBK			
$1a.$	$A \longrightarrow_N B$	:	$A, INFO_A, hash(A, k_A)$
$1b.$	$B \longrightarrow_N A$	:	$B, INFO_B, hash(B, k_B)$
$[A]$	$d_A := delay(k_A, T)$		
$2a.$	$A \longrightarrow_N B$	:	d_A
$[B]$	$d_B := delay(k_B, T)$		
$2b.$	$B \longrightarrow_N A$	:	d_B
$3a.$	$A \longrightarrow_N B$	:	k_A
$3b.$	$B \longrightarrow_N A$	:	k_B
$[A]$	$intime(k_A)$		
$[B]$	$intime(k_B)$		
$4.$	$A \longleftrightarrow_E B$	:	$digest(k_A \oplus k_B, INFOS)$

In other words, after each of the two parties is completely committed to the final value of the digest it sends the other one the delayed reveal of its hash key, and is only prepared to do the digest comparison if it has received the unencrypted hash key from the other *before* any party can have extracted the data from the respective d_X. It follows that in order to know whether the usual man in the middle attack will work, Eve either has to wait for $delay(k_A, T)$ and $delay(k_B, T)$ to open or to get A to send the open k_A and B to send k_B. If she does the first she will know if her attack would have succeeded earlier, but it will not now because the booleans tested by A and B before Message 4 will be false. If she takes the second approach she will have to have at the very least sent data representing d_A to B and d_B to A as a continuation of her man in the middle attack. However if she does, gets the values of k_A and k_B and abandons the attack, all A and B have to do is wait. After time T they can endeavour to open the delayed values and one of two things will happen:

- The ds they hold open successfully, are consistent with the hashes they hold, and A and B can compare s_A and s_B and find that they are different.

- The d do not open successfully or are not consistent with the Message 1 contents.

The second of these possibilities is unlikely, for if it happens then it would provide post-hoc evidence that the intruder was involved even if Eve had got lucky with the short strings: she would have had to have provided a d inconsistent with the Message 1 she sent to the same party *before* she knew if the two runs' short strings agreed.

In either case A and B can jointly deduce that an effort had been made to attack them. The revised protocol achieves the trichotomy above. Note that we can be confident that even if the *delay* function does not work at all, this protocol is still as secure as the original SHCBK presented above: for the original protocol fully reveals k_A and k_B at the points where the two delayed versions are revealed.

This approach readily extends to the group version of SHCBK of [18, 19])

The following is an auditable version of ZRTP. Note that we have ad to defer the release of g^{x_B} via hash commitment to make this possible.

Auditable ZRTP			
1.	$A \longrightarrow_N B$	:	$hash(g^{x_A})$
$[B]$	$d_B := delay(g^{k_B}, T)$		
2.	$B \longrightarrow_N A$	:	$d_B, hash(B, g^{X_B})$
3.	$A \longrightarrow_N B$	:	g^{X_A}
4.	$B \longrightarrow_N A$	:	g^{X_B}
$[B]$	$intime(g^{k_B})$		
5.	$A \longleftrightarrow_E B$	:	$shorthash(g^{x_A}, g^{x_B})$

The reason for including B's name in the hash in Message 2 is the same reason it is included in SHCBK: it prevents reflection attacks where A's own g^{X_A} is replayed to her in Message 2.

Finally we move on to the indirect binding Vaudenay protocol. To do this we must, in much the same way as in the previous case, both delay R_B and allow its eventual send to be checked against an earlier commitment. Because R_B is short the best way of doing this is using the same sort of *commit* construct as for R_A, and in that context we might as well add in any information B wants to authenticate to A (which could of course be null).

Auditable Vaudenay style protocol	
	$c_A \parallel dc_A := commit(INFO_A, R_A)$,
1.	$A \longrightarrow_N B : INFO_A, c_A$
	$c_B :=\parallel dc_B = commit(INFO_B, R_B)$,
	$dy_B := delay(R_B, T)$
2.	$B \longrightarrow_N A : INFO_B, c_B, dy_B$
3.	$A \longrightarrow_N B : dc_A$
$[B]$	$R_A := open(INFO_A, c_A, d_A)$
	B computes $R_A = open(INFO_A, c, d)$
4.	$B \longrightarrow_N A : dc_B$
$[A]$	$R_B := open(INFO_B, c_B, d_B)$
5.	$A \longleftrightarrow_E B : R_A \oplus R_B$

4 Options for time delay

The solution above depends on a construct that locks data away for a given period. It is recognised – see [1, 27], where many ideas similar to the ones presented below can be found – that there are two basic options for this

- The use of trusted third parties.
- Creating a computation that simply takes at least a given time to perform by any party.

Neither of these is ideal: trusted third parties are potentially corruptible and is hard to guarantee that no-one can perform some computation quickly while relatively low-power computers such as smartphones must nevertheless be able to do the same calculations rapidly enough to audit a failed run.

This seeming fragility is one very good reason why we have been careful, in our auditable protocols, not to rely on the *delay* construct to provide protocols' basic security. Of course the incentive for at attacker overcoming whatever delay mechanism is used is reduced if it is being used for auditability only.

We also remark that, in common with the traditional cryptographic one-way functions of hashing and public-key cryptography, it may well be necessary to salt a value that is being *delay*ed to prevent searching attacks by other parties. (This is also analogous to the nonce X used to prevent searching attacks against the c of commit schemes as referred to earlier.)

The most obvious way of creating the $delay(x,t)$ construct used in the last section appears to be using a trusted third party. Such a server S would perform the following service:

- When sent some data x encrypted under S's public key together with time t it will reply with a token y combined with a hash of (x,t).
- When sent y at least t beyond the point where it sent this reply, it reveals x.
- The values sent by S are signed by it.

In practice y might or might not contain the information S needs to recover the value of x without resorting to its own memory. That would affect whether the two instances of S used must be the same, or merely share keys with each other.

The usability of this, like the other forms of delay discussed below, will depend heavily on the context in which the HISP is being used. This particular one is unlikely to work well if Alice and Bob are likely to be cut off from any server when the protocol is run, but would be very suitable when they have to depend on the presence of a TTP for other reasons in their exchange. This would be true, for example, in the models of electronic transactions anticipated in [7].

Note that the form of time used in $delay(x,t)$ is relative: t is a delay from the present time, rather than an absolute time at which x will open. There is an advantage in this in that we would otherwise have to worry about how accurate the knowledge of the present time in each node might be. However the work required of the TTP above might be considerable if it was serving a lot of clients. The following alternative approach, similar alleviates this.

We create a server TL that issues a series of asymmetric "time-lock" keys, each labelled with the time it was created and the time it will open, each signed by TL. When the appointed times for opening these keys, come round, TL issues them. So all this TTP has to do is post a series of signed keys.

For example our server might issue one key every 10 seconds, with a delay of one minute before the counterpart was issued. The key pairs might then be made available online for several days to enable auditing.

Agents can still implement $delay(x,t)$ using this service, but only if they can bound the divergence between their own system time and that employed by TL. This knowledge will enable them to pick an already posted key that will not be opened by TL for at least t units of time, and encrypt x under it.

The final option is to do without a TTP and make anyone who wants to open $delay(x,t)$ do a lot of work. Given that we expect these objects to be created a lot more often than opened it makes sense to want a version that is cheap to create. Given our application we want a method that will allow the sort of device running our protocols to be able (as part of the auditing process) to be able to open such an object in a small multiple (say 10) of the time it will take an attacker with essentially unbounded computing resources to do this. As discussed in [1], this means that the computation must be sequential, in the sense that it is impossible to parallelise. Several potential schemes are given there, based on a long sequence of operations. One possibility is

- A large prime p of the form $3m+2$ is chosen[2], the number of digits determining a time delay t. $delay(x,t) = x^3 \, mod \, p$ is fast to compute, but to uncover x from this we have to compute $x^d \, mod \, p$ where d is chosen so that $p-1$ divides $3d-1$. This will typically take $(log_2 \, p)/2$

[2]These are exactly the primes in which cubing x^3 is invertible. Squaring is not invertible for primes other than 2.

times longer to compute, namely a computational advantage proportional to the number of digits.[3] To achieve a sufficient delay it may be necessary to make p very large indeed; as an alternative we could repeat this operation with several increasing primes in turn. The primes in this example could be published and used by all nodes.

While this computation has at its core the need to repeatedly square numbers in some modulus, it is hard to exclude, particularly in cases where pre-computations can be done for a particular modulus, special purpose hardware being used to speed up the individual multiplications, or there may be better algorithms employed than are widely known.

In any case it is likely to be very hard to *prove* that a particular calculation will certainly take a given time t, particularly when for auditing purposes we want relatively low power devices to be able to do the same calculation in a relatively small multiple of t. Of course the smaller we can make t, which will be determined by our confidence in the infrastructure implementing the protocol, the larger.

Thus each of our alternative implementations of $delay(x, t)$ is subject either to the corruption of TTPs or to imponderables about the limits of computation and parallelisability. In the author's mind it is this which makes it highly advisable to use the approach to auditability we have, in which the basic security of the protocol is not compromised by it.

5 Conclusions

We have shown how a variety of protocols can be made auditable by the addition of additional data fields and/or messages. All of these transformations involve the addition of time-dependent data to replace or supplement the mechanisms already present to commit agents to data before they know it. We have shown that this can be done straightforwardly if we can rely sufficiently on the security of the *delay* mechanism, and in a way that does not risk the security of the original protocols otherwise.

Finally we have given some examples of how *delay* might be implemented, observing that what method is preferable will depend on context. Indeed, one can imagine that a single protocol implementation might well use different *delay*s as alternatives (e.g. depending on the availability of TTPs.)

One of the assumptions we have made in this paper is that when a message is received by a node over $\longrightarrow_N$ that is not consistent with the protocol, an attack is diagnosed. In order to make this reasonable, protocol implementers should ensure that accidental corruptions, mis-deliveries and abbreviations of messages are not accepted as "messages" at all. This will typically be achieved by using very explicit message formatting, e.g. in XML, and by including integrity information such as hashing to ensure that accidental issues relating to messages sent between trustworthy parties will essentially *never* lead to a node thinking it has received a message that was not sent to it in the form that was sent. Such precautions will not prevent an attacker from sending messages that look real, but should mean that agents can reject accidentally corrupted messages.

All the protocols we have considered so far in this paper have three properties:

- The oob channel is used simply for comparing two short strings.
- This comparison takes place at the end of the protocol.

[3]The calculation of x^3 will clearly take more time, the more digits there are. Note that there are multiplication algorithms faster than the usual "schoolbook" one that can be expected to give significant advantages when p is very long.

- Each side of the protocol introduces something that randomises these strings.

Other protocols that have these properties and can easily be made auditable using the techniques set out here are Bluetooth V2 [2],

References

[1] Time-Lock Encryption. `http://www.gwern.net/Self-decrypting`. 2011.

[2] *Simple Pairing Whitepaper*, Bluetooth Special Interest Group, 2006. See: www.bluetooth.com/NR/rdonlyres/ 0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/SimplePairing_WP_V10r00.pdf

[3] Wikipedia article on ZRTP: `https://en.wikipedia.org/wiki/ZRTP`

[4] D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, **46**(2), 203-213. 1999.

[5] M. Čagalj, S. Čapkun and J. Hubaux, Key agreement in peer-to-peer wireless networks, in: *Proceedings of the IEEE Special Issue on Security and Cryptography* **94**(2) (2006), A. Mazzeo, ed., 467-478.

[6] J.L. Carter and M.N. Wegman, Universal classes of hash functions, *Journal of Computer and System Sciences* **18**(2) (1979), 143-154.

[7] Bangdao Chen and A.W. Roscoe. Mobile electronic identity: securing payment on mobile phones. *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication* (pp. 22-37). Springer Berlin Heidelberg. 2011,

[8] D. Dolev and A.C. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, **29**(2), 198-208. 1983.

[9] C. Gehrmann, C. Mitchell and K. Nyberg, Manual authentication for wireless devices, *RSA Cryptobytes*, **7**(1) (2004), 29-37.

[10] C. Gehrmann and K. Nyberg, Security in personal area networks, in: *Security for Mobility*, C.J. Mitchell, ed., IEE, London, 2004, pp. 191-230.

[11] ISO/IEC 9798-6, C. Mitchell, ed., 2003, *Information technology – Security techniques – Entity authentication – Part 6: Mechanisms using manual data transfer.*

[12] M.T. Goodrich, M. Sirivianos, J. Solis, G. Tsudik and E. Uzun, Loud and clear: Human-verifiable authentication based on audio, in: *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006)*, 2006, pp. 10-33.

[13] J.-H. Hoepman, Ephemeral pairing on anonymous networks, in: *Proceedings of the 2nd International Conference on Security in Pervasive Computing (SPC 2005)*, Lecture Notes in Computer Science, Vol. 3450, D. Hutter and M. Ullmann, eds., Springer, 2005, pp. 101-116.

[14] J.-H. Hoepman, Ephemeral pairing problem, in: *Proceeding of the 8th International Conference on Financial Cryptography*, Lecture Notes in Computer Science, Vol. 3110, A. Juels, ed., Springer, 2004, pp. 212-226.

[15] S. Laur and K. Nyberg, Efficient mutual data authentication using manually authenticated strings, in: *Proceedings of the 5th International Conference on Cryptology and Network Security (CANS 2006)*, Lecture Notes in Computer Science, Vol. 4301, D. Pointcheval, ed., Springer, 2006, pp. 90-107.

[16] S. Laur, N. Asokan and K. Nyberg, Efficient mutual data authentication using manually authenticated strings: Extended version, in: *Cryptology ePrint Archive*, Report 2005/424, 2006.

[17] A. Mashatan and D.R. Stinson, Non-interactive two-channel message authentication based on hybrid-collision resistant hash functions, in: *IET Information Security* **1**(3) (2007), 111-118.

[18] L.H. Nguyen and A.W. Roscoe, Efficient group authentication protocol based on human interaction, in: *Proceedings of the Joint Workshop on Foundation of Computer Security and Automated Reasoning Protocol Security Analysis (FCS-ARSPA 2006)*, 2006, pp. 9-31.

[19] L.H. Nguyen and A.W. Roscoe, Authenticating ad-hoc networks by comparison of short digests, *Information and Computation* **206**(2-4) (2008), 250-271.

[20] L.H. Nguyen and A.W. Roscoe, Separating two roles of hashing in one-way message authentication, in: *Proceedings of the Joint Workshop on Foundations of Computer Security, Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (FCS-ARSPA-WITS 2008)*, 2008, pp. 195-210.

[21] L.H. Nguyen and A.W. Roscoe, Authentication protocols based on low-bandwidth unspoofable channels: a comparative survey. *Journal of Computer Security*, **19**(1), 139-201. 2011

[22] L.H. Nguyen and A.W. Roscoe, Short-output universal hash functions and their use in fast and secure data authentication. In Fast Software Encryption (pp. 326-345). Springer 2012.

[23] L.H. Nguyen, *Authentication protocols in pervasive computing*, D.Phil. Thesis, University of Oxford, 2010.

[24] ISO/IEC 9798-6 (revision), L.H. Nguyen, ed., 2010, *Information Technology – Security Techniques – Entity authentication – Part 6: Mechanisms using manual data transfer.*

[25] S. Pasini and S. Vaudenay, SAS-based authenticated key agreement, in: *Proceedings of the 9th International Conference on Theory and Practice of Public-Key Cryptography (PKC 2006)*, Lecture Notes in Computer Science, Vol. 3958, M. Yung, Y. Dodis, A. Kiayias and T. Malkin, eds., Springer, 2006, pp. 395-409.

[26] S. Pasini and S. Vaudenay, An optimal non-interactive message authentication protocol, in: *Proceedings of the Cryptographers' Track at the RSA Conference 2006 on Topics in Cryptology*, Lecture Notes in Computer Science, Vol. 3860, D. Pointcheval, ed., Springer, 2006, pp. 280-294.

[27] R.L. Rivest, A. Shamir and D.A. Wagner. Time-lock puzzles and timed-release crypto. 1996. `http://bitsavers.trailing-edge.com/pdf/mit/lcs/tr/MIT-LCS-TR-684.pdf`

[28] A.W. Roscoe, Human-centred computer security, 2005. See: http://web.comlab.ox.ac.uk/oucl/work/bill.roscoe/publications/113.pdf.

[29] F. Stajano and R. Anderson, The resurrecting duckling: Security issues for ad-hoc wireless networks, in: *Proceedings of the 7th International Workshop on Security Protocols*, Lecture Notes in Computer Science, Vol. 1796, B. Christianson, B. Crispo, J. A. Malcolm and M. Roe, eds., Springer, 1999, pp. 172-194.

[30] F. Stajano and R. Anderson, The cocaine auction protocol: on the power of anonymous broadcast, in: *Proceedings of the 3rd International Workshop on Information Hiding*, Lecture Notes in Computer Science, Vol. 1768, A. Pfitzmann, ed., Springer, 2000, pp. 434-447.

[31] J. Valkonen, N. Asokan and K. Nyberg. Ad Hoc Security Associations for Groups. In *Proceedings of the Third European Workshop on Security and Privacy in Ad hoc and Sensor Networks*, Hamburg, Germany, September 2006. Volume 4357 of Lecture Notes in Computer Science, Springer.

[32] S. Vaudenay, Secure communications over insecure channels based on short authenticated strings, in: *Advances in Cryptology - Crypto 2005*, Lecture Notes in Computer Science, Vol. 3621, V. Shoup, ed., Springer, 2005, pp. 309-326.

[33] P. Zimmerman. ZRTP. `https://tools.ietf.org/html/draft-zimmermann-avt-zrtp-22`. 2010.