Available online at www.sciencedirect.com
ScienceDirect
Information and Computation 206 (2008) 250–271
Information and Computation
www.elsevier.com/locate/ic

Authenticating ad hoc networks by comparison of short digests

L.H. Nguyen *, A.W. Roscoe

Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, UK

Received 10 November 2006; revised 1 May 2007
Available online 28 November 2007

Abstract

We show how to design secure authentication protocols for a non-standard class of scenarios. In these authentication is not bootstrapped from a PKI, shared secrets or trusted third parties, but rather using a minimum of work by human user(s) implementing the low-band width unspoofable channels between them. We develop both pairwise and group protocols which are essentially optimal in human effort and, given that, computation. We compare our protocols with recent pairwise protocols proposed by, for example, Hoepman and Vaudenay. We introduce and analyse a new cryptographic primitive—a digest function—that is closely related to short-output universal hash functions.
© 2007 Elsevier Inc. All rights reserved.

1. Introduction

Imagine that a group of people come together and agree that they want to transfer data between them securely, meaning that they want it to be secret and of authenticated origin. They all have some pieces of computing hardware (e.g., a mobile phone or a PDA). Unfortunately none of them knows the unique name of any of the others' equipment, and in any case there is no PKI which encompasses them all. How can they achieve their goal in the context that their machines are connected by an insecure network (whether created by WIFI, the internet, telephony, or a mixture of these)?

The conventional answer to this question would be that this goal is unachievable, since it is impossible to prevent some impostor I playing the *man-in-the-middle* between the participants. However a little creative thinking can easily solve the problem: if each person tells the others (using human conversation) a public key for his or her machine, they can then use something like the Needham–Schroeder–Lowe protocol [27] (over the insecure network) to establish secure and authenticated communication. (If there are more than two participants then they would either have to adapt that protocol or to use it multiple times.) They will have bypassed the intruder for the crucial step of exchanging electronic identities.

The problem with that approach is that it requires too much human effort for practical purposes. In this paper, we introduce some much better methods. The danger of combinatorial attacks means that we require

* Corresponding author.
Email addresses: Long.Nguyen@comlab.ox.ac.uk (L.H. Nguyen), Bill.Roscoe@comlab.ox.ac.uk (A.W. Roscoe).

0890-5401/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2007.07.010

more subtle analysis than with traditional authentication protocols, but develop two principles and the concept of a *digest* function that together lead to the development of efficient protocols.

Our protocols were, in the main, developed before we were aware of several other recent (pairwise) protocols reported in [21,22,37,8]. We compare ours against these others, comparing both the amount of human and computational resources required by the different styles.

We introduce a number of different protocols in this paper that are suitable for many different sizes and types of group, all the way from pairwise connection between the systems of people who completely trust each other, to a lecture theatre full or more, for example. This, and the range of potential implementation technologies, mean that in this paper we largely abstract away the details that are not immediately important to security. We also have imagined that there is a preliminary and insecure group set-up protocol (implementation dependent) that is run either before or simultaneously to the first messages of the secure protocol.

This paper is organised as follows: in the next section we give some arguments as to why this class of protocols is important, show how protocols for this type of scenario can be vulnerable to combinatorial attacks, and analyse how others have solved this problem. In Section 3 we introduce a class of efficient group formation protocols that rely on trust and which seem particularly appropriate for a user's equipment to set up secure communications with a group of simple devices. In Section 4 we show how to build groups securely even when corrupt participants are present in the group, using extra computation to dispense with the need for trust. We then analyse the requirement of the digest function our protocols use and suggest some implementations. Finally we analyse the relative efficiency of our protocols and those of [21,22,37,8], and look to future verification work.

1.1. Contribution

The original contribution of this paper is first represented by the protocols we introduce, which efficiently solve a problem that we believe to be of great practical importance. The second main contribution is the creation of efficiently computable *digest* functions that share some similarities to short-output *uniform families of hash functions* and *universal hash functions* as originally defined in [38,11], that can be considered perfect for cryptographic purposes.

2. Background

The use of public key infrastructures, trusted third parties, or shared secrets will be anywhere from inconvenient to impossible in emerging pervasive computing applications. Equally, even when these are present they frequently only identify nodes by their *ID* fields—often completely inappropriate in the world of lightweight, human-driven communication. We are much more likely to identify the intended participants by some aspect of their context: for example a node's human user or its physical nature and position.

This has been recognised in the development and popularity of the Bluetooth protocol, though this has been recognised as having significant security flaws. Unfortunately it has the weakness of only being usable in (usually local) situations where there is a secret shared password, and is subject to severe off-line password guessing attacks [23].

It has been widely realised that it is impossible to bootstrap security from nothing. Nevertheless, as we have discussed above, it is necessary to be able to bootstrap it from minimal assumptions. So what is it reasonable to assume exists prior to an attempt to acquire a high-quality security? There have been (at least) two separate approaches to this. One is to assume that the pair of parties, who are seeking to exchange a strong secret key, already share some short or low-entropy secret such as a password. The other assumes the existence of a low-bandwidth empirical channels that are not susceptible to spoofing, though they are liable to be overheard or blocked by the intruder.[1] Based on either assumption, the parties can agree on a strong shared secret from scratch.

[1] In the presence of blocking and overhearing, and the absence of spoofing, there is a further question: is the channel *delayable*. We will assume that these channels are not substantially delayable in the sense that a message blocked in one protocol run can be delayed until a subsequent one.

The first method, which is based on possessing a shared password, has been studied extensively in the last decade. Bluetooth itself is such a protocol, but a weak one as discussed above. The community then came up with various formal frameworks presented in [3,4,5,9,10] by Bellare, Pointcheval, Rogaway, Canetti, and Krawczyk and others that focus on preventing off-line dictionary attacks that Bluetooth allows in [23]. These ensure that the only way that an attacker can find out whether his random guess of the password is correct or not is by interacting with the legitimate player. Typically these reduce the probability of a successful attack to that of guessing the password correctly in a single trial.

The need to keep the password secret in these precludes their use in scenarios subject to eavesdropping, including most remote ones since the password cannot be transmitted without pre-existing security that would render the protocol redundant.

Taking a different approach that makes use of the empirical channel, in [12,13,14,15] Roscoe, Creese, Goldsmith, Zakiuddin and others attempted to form a secure network for both two-party and multi-party scenarios. However, as shown below, their scheme is vulnerable to a combinatorial attack, related to the birthday paradox. There has been also other work that concentrates on two-party protocols, presented in [21,22,37,8,2,16] by Hoepman, Vaudenay, Balfanz, Gehrmann, Mitchell, Nyberg and others.

2.1. Combinatorial attack on a group formation protocol

The following protocol was originally proposed in [14], where *shorthash* and *longhash* are hash functions with short and long outputs, respectively.

Group protocol of Creese, Roscoe, and others
1. $\forall A \longrightarrow_N \forall A' : A, Pk_A, N_A$
2. $\forall A \longrightarrow_N \forall A' : \{\text{all Messages } 1, N'_A\}_{Pk_{A'}}$
3*a*. A displays : $shorthash(\{\text{all Messages } 2^d\})$, number of processes
3*b*. $\forall A \longrightarrow_E \forall A'$: users compare hashes and check numbers
4. $\forall A \longrightarrow_N \forall A' : longhash(\{\text{all Messages } 2^d\})$

Here, $\forall A$ means that a message is sent or received by all parties in the group **G** who attempt to achieve a secure link between their laptops or PDAs. Pk_A stands for an uncertificated public key for A. The superscript Message 2^d represents the decrypted content of Message 2. In addition, Message 4 in this protocol does not add any extra security to the scheme, its presence only aims to provide a confirmation of the shared secret information.

The protocol uses two types of channel:

- $\longrightarrow_N$, the normal Dolev–Yao network where all messages transmitted between the laptops in this channel can be overheard, deleted or modified by the intruder.
- $\longrightarrow_E$ indicates the low-bandwidth empirical channel, typically implemented by human users, which is similar to the authentic channel used in [21,22], and not susceptible to spoofing.

This protocol introduced, implicitly, the first of two principles which underlie the new protocols we will be describing in this paper:

P1 Make all the parties who are intended to be part of a protocol run empirically agree a short-output hash or *digest*[2] of a complete description of the run.

In all the protocols we introduce in this paper, the "complete description of the run" is identified with *INFOS*, the collection of all the information that any member of the group wishes to have authenticated to it: the concatenation of pairs of the form $(A, INFO_A)$. Once the agreement required in **P1** has occurred then, unless there is a hash or digest anomaly—different nodes in the group computing the same hash value or digest from different antecedents—then all the parties agree on all the data transmitted during the protocol.

[2] A digest function here means a type of short-output hash function that we will formally specify in Section 3 and analyse in Section 5.

We can state the impact of **P1** in the following theorem. In this, A is the name of a node that it uses electronically, whereas α and β are the identities of nodes as perceived in the human world, formally through the outgoing empirical channels they have. An identity of this second sort might be the human user of the node or some composition of position and nature. One of the main purposes of our protocols will be binding "logical identities" such as A to "empirical identities" such as α.

Theorem 1

(1) Suppose that, in a given protocol, a pair α and β of trustworthy nodes empirically agree on a hash value that node α has computed as hash(INFOS), where INFOS includes $(A, INFO_A)$ and no other binding to the name A. Then if β fails to bind $INFO_A$ to A then there is a hash anomaly.

(2) Suppose furthermore that α would not have agreed the value unless its own binding was present in INFOS, and that all other bindings than $(A, INFO_A)$ are inconsistent with them being associated with α, then in the absence of a hash anomaly β can reliably bind $(A, INFO_A)$ to α.

Proof. The proof of part (a) is obvious.

The assumption in Part (b) is that each $(A, INFO_A)$ contains information that allows the human user to correlate it with potential empirical nodes such as α.[3] The proof of part (b) then follows since we know that α has announced some logical identity for itself, and none other than A are possible. Of course, if the information attached to A is also inconsistent with α, then β can deduce that α is not reliable. □

Since all our protocols will make use of **P1** and this theorem, we need not worry further about the distinction between logical and empirical nodes: these allow the creation of reliable methods for binding one to the other.

The question that remains with the protocol above is, therefore, whether a hash anomaly is possible. Since the short hash values in Messages 3a are compared manually by human effort, and not by computer, the length of the short hash can only be up to a few digits or characters. It turns out that it is not hard for an intruder, in a limited amount of time, to search for a collision that might cause the parties to agree on different secret information; and thereby to force an anomaly in the sense described above.

The attack can be described as follows: the intruder will run two parallel sessions with two subsets S_1 and S_2 that partition the group **G** of N parties. During the two parallel runs, the intruder impersonates all parties of subset S_1, by modifying messages sent from subset S_1 with different public keys where she knows the corresponding secret keys, to talk to parties in the other subset, S_2, and vice versa. Therefore, any one in the group **G** is still thinking that s/he is running the protocol with the other $(N-1)$ parties. The intruder only allows messages to be passed unaltered within each subset, but when a message is intended to go across the boundary between the two subsets its content will be changed appropriately.

If $S_1 = \{A, B, C\}$, $S_2 = \{D, E\}$, and $\{N'_A, N'_B, N'_C, N'_D, N'_E\}$ are the original nonces randomly created by all the parties in Messages 2 then the adversary creates shadow copies of these nodes A'–E' as shown in Fig. 1, and has to generate corresponding nonces $\{N''_A, N''_B, N''_C, N''_D, N''_E\}$ for their Messages 2. It seeks to choose these values so that

$$shorthash(N'_A, N'_B, N'_C, N''_D, N''_E) = shorthash(N''_A, N''_B, N''_C, N'_D, N'_E)$$

This can be done by picking the nonces N''_A, N''_B, N''_D randomly and then, based on the birthday paradox, the intruder can search for values of N''_C and N''_E such that the hashes come out to be equal to each other. This search, shown in the figure, can be expected to take time proportional to $\sqrt{H}$, where H is the number of different short hash values.

If we are to attain our goal of optimising the amount of security obtained from a given amount of empirical communication (essentially hash width) we need to eliminate not only birthday attacks like this one but also

[3] This explicitly means that each $INFO_B$ contains information that may indicate that B is likely to be associated with an empirical node β, or may indicate that it cannot be β if β is trustworthy. For example: if $INFO_B$ says that its empirical user is *Bill*, then it is likely but not certain it is the *Bill* we want to connect to, but if the person we want to connect to is called *Long*, then we know that B does not belong to *Long*.

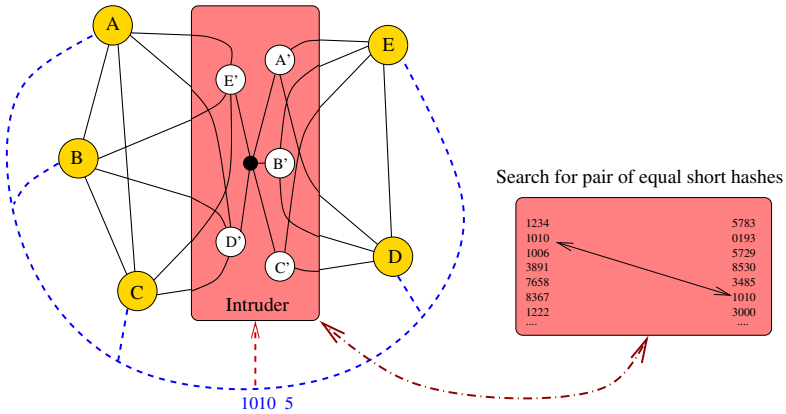

Fig. 1. Birthday attack on a short hash, where the continuous and dashed lines indicate the Dolev–Yao and empirical channels, respectively.

ones in which the intruder is able to achieve something if a basic combinatorial search for a value v such that $hash(v) = x$ succeeds, where x is fixed. Normally, where cryptographic hashes are used, we choose them long enough so that this type of attack is pointless (the property of *collision resistance*); we are going to have to devise protocols that offer the adversary no opportunity to carry out searches of the types discussed above.

This example also illustrates that in creating and analysing protocols that deliver strong security for a minimum of empirical communication, we will need to go beyond the style familiar from the Dolev–Yao model where all answers tend to be clear "secure" or "insecure", but is likely to lead to subtle quantitative reasoning.

2.2. Other peer-to-peer key exchange protocols

In [2,32], Balfanz, Pasini and Vaudenay proposed two protocols for key exchange. Both of the schemes, however, require a large number of bits to be communicated over the authenticated channel and to be compared manually by using human effort. Taking a further step, in [21,22,37,8], Hoepman, Vaudenay, and Čagalj proposed protocols that can get around the problem of the bandwidth of the authentic channel. They all share many similarities: for example, all of them concentrate on the case where there are two parties in a *peer-to-peer* network. In addition, they contain the idea of pre-committing two parties to some random secrets by sending the corresponding cryptographic hash, in [21,22], or the output of the commitment scheme, in [37,8], to each other at the start of the protocol.

In [21,22], Hoepman required each party to compute and manually compare two short authenticated strings that are transmitted over the empirical channel:

Hoepman pairwise two-way authentication protocol
1. $A \longrightarrow_N B : longhash(g^{x_A})$
1′. $B \longrightarrow_N A : longhash(g^{x_B})$
Where x_Y is randomly picked by Y
2. $A \longrightarrow_E B : shorthash(g^{x_A})$
2′. $B \longrightarrow_E A : shorthash(g^{x_B})$
3. $A \longrightarrow_N B : g^{x_A}$
3′. $B \longrightarrow_N A : g^{x_B}$
A and B recompute to verify the long and short hashes.
A and B then share the key $k = g^{x_A x_B}$

This is not optimal in the amount of work required by the humans implementing the empirical channel, since the same amount of security (i.e., improbability of a successful attack) can be obtained in several ways by them comparing a single string of the same length. We will give two methods for doing this later; a further one has been devised by Vaudenay [37], and adapted by Čagalj et al. [8]. Vaudenay and Čagalj require a commitment

scheme[4] that is at least as secure as a standard cryptographic hash function.[5] By this we mean that it must be computationally infeasible to find, with greater than infinitesimal probability, collisions, or inverses to the "commit" values. It should therefore be assumed that these values are as hard to compute as, and have as many bits as, a strong cryptographic hash.

Vaudenay, in common with several other authors in this area, writes the juxtaposition of two pieces x and y of data as $x \parallel y$: therefore we use this notation when discussing their protocols. We do not use this notation in our own protocols because of the potential for confusing this notation with its common use as a parallel operator.

Vaudenay pairwise one-way authentication protocol
1. $A \longrightarrow_N B : INFO_A, c$
Where $c \parallel d = commit(INFO_A, R_A)$,
R_A is a short random nonce of A.
2. $B \longrightarrow_N A : R_B$
3. $A \longrightarrow_N B : d$
B computes $R_A = open(INFO_A, c, d)$
4. $A \longrightarrow_E B : R_A \oplus R_B$
B verifies the correctness of $R_A \oplus R_B$

Note that the final value compared in this protocol is the XOR of the short entropies devised by A and B. In particular it does not depend functionally on the information $INFO_A$ being sent; in other words it does not follow our principle **P1**. The guarantee of authenticity of $INFO_A$ that this protocol delivers is as a consequence of:

- The fact that this exchange guarantees the value for R_A that B has discovered from the commit scheme is the one that A intended.
- The way the commit scheme has strongly bound $INFO_A$ to R_A at a point where R_A is itself unknown to any attacker.

We will see in Section 6 that this indirect binding of $INFO_A$ to the final agreement makes this protocol relatively expensive relative to others we will introduce.

In this paper, we shall extend the idea of [12,13,14,15] in constructing an arbitrary-sized secure network, but without the trustworthiness of the entire network. We also believe that the degree of security obtained is essentially optimal for the amount of empirical (human) communication required.

In the mean time, we also try to reduce disadvantages of [21,22,37,8] with respect to efficiency when it comes to implementing the commitment scheme and computing the long and short hashes, or digests.

3. Some protocols for bootstrapping groups

We will introduce our protocols in order they were discovered. The ones presented in this section are based on one discovered by Roscoe in June 2005.[6]

All the protocols we propose in this paper use the principle **P1** to bind $(A, INFO_A)$ to empirical identities. It follows that, to achieve this, all we have to do is to avoid hash or digest anomalies. Our reasoning about these protocols below is mainly confined to this as Theorem 1 then establishes the main property we want. In particular, as stated earlier, this means we do not need to reason about the relationship between empirical and

[4] The commitment scheme used in this protocol consists of two functions. $c \parallel d = commit(R_A, INFO_A)$ and $INFO_A \parallel R_A = open(c \parallel d)$. The combination of two pieces of data will frequently be written $x \parallel y$, this will be synonymous with the ordered pair (x, y). R_A is a short random nonce generated secretly by party A. A intends to bind R_A and $INFO_A$ together without revealing R_A by publishing the commitment c. Eventually sending d (the *decommitment*) reveals R_A, and binds this value firmly to $INFO_A$ in the eyes of the receiver. As R_A is a short random nonce, the party A needs to extend R_A by adding extra random bits so that the security of the scheme in term of both *binding* and *hiding* is equivalent to a standard cryptographic hash function.

[5] We note that there is a lack of explicitness in the specification of the commitment scheme in [37], since the security specification there fails to bind it to the Message m, as was obviously intended. The definition there is satisfied by $commit(m, r) = Hash(N, r) = c$, and $open(m, c) = (N, r)$, for N a fresh nonce.

[6] In the notation of this section, that was HCBK3.

logical nodes, though of course the $(A, INFO_A)$ have to contain sufficient information to allow the mechanisms assumed by Theorem 1 to occur.

There is a slightly grey area for protocols building *groups* of more than 2: should we or should we not be content if the presence of a corrupt party in a group means that communications that result between trustworthy members of the group are themselves compromised? In some of the circumstances where we may wish to use ad hoc group formation protocols it would be much better if the protocols were tolerant of corrupt members. We will therefore be careful about our assumptions on this front.

It is obvious that any protocol which creates a shared secret is at least partially compromised by the presence of a corrupt participant. However protocols which merely authenticate public-key-like information to nodes are not globally compromised in the same way: they could be said to be establishing a *local PKI*.

In this section we will assume that there is one participant I in the protocol whom all agree is trustworthy. This could be because all participants are known to be trustworthy, because I has some special status amongst them, or because I is the only one requiring authentication. I will be called the "initiator", and the other nodes will be termed "slaves". We will first give a protocol that is designed for the case where there are empirical channels both from all nodes to I and from I to all nodes. (It will be obvious that in some aspects the protocol might work more naturally if there were empirical channels between all nodes.)

A crucial component of this and subsequent protocols is a *digest* function that takes two arguments: $digest(k, M)$. The first is a key that we will try to ensure varies randomly over a large space. The second is the data that is being "digested", typically the aggregate of all the information that is published by the nodes in the protocol, as suggested by Theorem 1. The width of the digest b (i.e., the number of bits it produces) will always be the number we want to communicate along the empirical channel. Therefore we will seek to minimise b subject to obtaining a given measure of security. Of course, avoiding hash/digest anomalies will always be something that we can only do up to a certain probability, since a pure guess will have a 1 in H chance, where $H = 2^b$. What we want to do is to ensure that as the key k varies:

(1) $digest(k, M)$ is uniformly distributed for any fixed M.
(2) And for any fixed θ and $M \neq M'$:

$$\Pr\big(digest(k, M) = digest(k \oplus \theta, M')\big) \leqslant \frac{1}{2^b}$$

The rationale for these two specifications will become apparent when we analyse the protocols we introduce below.

We will assume for the time being that we have a digest function meeting this specification, or at least one that meets it up to some notion of cryptographic certainty. We will show how to implement such a function in Section 5.

We now describe our first protocol that avoids hash/digest anomalies. In the following S represents a typical slave node, A a typical node (either I or S), *longhash* is a strongly collision-resistant and inversion-resistant hash function and *digest* is a digest function as described above. $init(I, A)$ is *true* if $I = A$ and *false* otherwise.

Hash Commitment Before Knowledge, HCBK protocol		
0.	$I \longrightarrow_N \forall S$	: I
1.	$\forall A \longrightarrow_N \forall A'$	: $(A, INFO_A)$
2a.	$I \longrightarrow_N \forall S$	: $longhash(k_I)$
2b.	$\forall S \longrightarrow_E I$	: committed
3.	$I \longrightarrow_N \forall S$	: k_I
4a.	$\forall A$ displays	: $digest(k_I, INFOS), init(I, A)$
4b.	$\forall A \longrightarrow_E \forall A'$	: Users compare and check presence of I

The meanings of these messages are as follows:

- **Message 1** publishes the information that all the nodes want to have attached to them, via the insecure channel. Therefore they do not know upon receiving it that it is accurate.
- **Message 2a** has I devise a key k_I with sufficient entropy that $longhash(k_I)$, which it publishes here, has no more than an infinitesimal likelihood of any combinatorial attack on it succeeding.

- **Message 2b** has all the slaves communicate to I over the empirical channel[7] that they have received Message 2a and are therefore *committed* to their final digest value (though none of them know it yet).
- **Message 3** has I publish the key k_I after it has received commitments from all members of the group over the empirical channels. All slaves now have the duty to check if the values of Messages 2a and 3 are consistent.
- **Message 4a** has all the nodes compute what should be the same digest value.
- **Message 4b** has them compare these values: this could be done either through the single point of contact at I or more generally. Once a node knows that all have agreed this value it has completed the protocol and can enter group mode. It also guarantees that one of the nodes doing the agreeing has been playing the initiator role.

3.1. HCBK protocol analysis

Thanks to Theorem 1, all we have to show is that our protocol prevents hash/digest anomalies.

(1) Digest anomalies are not impossible, since the intruder can partition **G** into two parts, and feed both of them different sets of values. It would then act as an empirically silent "initiator" in one of these subgroups. Picking a random value for the key will give the intruder atmost a 2^{-b} chance of the two digest values agreeing, thanks to the specification of the digest function. Our hope is that it is impossible for an attacker to have a better chance than this. To demonstrate this we analyse the positions the various nodes are in when they first become committed to their final digest value.

(2) I is committed when, following its acceptance of Messages 1, it creates the key k_I. A slave S is effectively committed once it has accepted Message 2a, even though at that stage it cannot know what the digest value is. For it has all information other than k_I, and it has $longhash(k_I)$, meaning that there is no better than an infinitesimal chance that it will accept a different k'_I in Message 3.
Thanks to the first part of the definition of a digest function, no party other than the initiator can know the final digest value it will use with any certainty at all: as far as they are concerned this value is still drawn from a uniform distribution over all digest values.

(3) So let us examine the state the network is in just before I publishes k_I in Message 3. The trustworthy node I is the only one that actually knows enough to compute the final digest—in particular no intruder can know the digest value d_I that I will compute in Message 4a. Furthermore, I knows—and therefore we know—that each slave S has been committed to some final digest value d_S. Some or all of the d_S may be different from d_I, and even equal ones may be based on different antecedents. But since all of the slave nodes S know the second argument $INFOS_S$ to their use of *digest* before the intruder knows k_I, it follows from the specification above that, unless $INFOS_S = INFOS_I$, the probability that $d_I = d_S$ is 2^{-b}.

(4) It follows that if, in Message 4, the various nodes go on to compare precisely these values and two of them have different values for *INFOS*, no matter what the intruder might have done, it will give him no better than the 2^{-b} chance that we have aimed for of them digesting to the same value.

(5) There is still one potential avenue of attack open: can the intruder change the mind of one or more participants about the final digest value so that it equals the others? The only way it could do this would be to make them abandon this run and bring them to the point in a subsequent run where they are ready to agree the final digest.
This would be impossible with the initiator. I is the final determiner of its own digest value by constructing k_I. So re-starting it would not give greater than 2^{-b} chance of achieving any particular value. Also, and conclusively, the initiator expects to get empirical signals in Message 2b from the slaves, and these would not be available from the slaves.

[7] The commitments must be transmitted over the unforgeable empirical channel because if it were not the case then the intruders could stand in the middle of the network to block all Messages 1 and 2 sent between the initiator and the slaves, and to fake the commitment signals from these slaves to the initiator. Once I hears the faked commitment signals, it then reveals the secret hash key k_I to the adversary, who now can start another run with all the slaves, the adversary will play the role of the initiator in the second run, but because it knows the hash key, and therefore it can constructively manipulate the information in Messages 1 and 2 for his own purposes such that the final digest values of both the slaves in the second run and the real initiator in the first run come out to be the same, even though they have different antecedents. This attack is similar to the one described in details in Appendix.

On the other hand, if a slave S could be re-started after d_I was known, then the intruder could perform a combinatorial search for a value k'_I which would yield the digest value d_I (with the combination of $INFO'_A$ of which he wants to persuade S), then a potential attack is open provided the second empirical Message 2b from S to I can be blocked. This would lead to an attack.[8] We therefore make the following specification for the implementation of the protocol:

> *The implementation must be designed so that agreement is impossible between final digest values other than those whose commitment has been signalled by the Messages 2b that I received.*

The most obvious way of achieving this is via timing limits: an upper bound on the time between I sending Message 3 and agreeing Message 4, and a lower bound between I receiving a Message 2b from S and S sending another Message 2b. One could also use run numbers that are included in empirical communications, but of course that would add to the empirical effort.

On the assumption that the above is achieved, we conclude that the nodes will never seek to agree final digest values to which they were committed later than the issue of Message 3 by I. Therefore our protocol achieves its goal of limiting the chance of a successful attack to at most 2^{-b}.

3.2. Modified versions of HCBK

We will call the above protocol HCBK1, standing for *Hash Commitment Before Knowledge*, the principle on which it works. Recall its goal: to agree a set of information of the form $\{(A, INFO_A) \mid A \in \mathbf{G}\}$ amongst the members of $\mathbf{G}$, and hence authenticate each such $INFO_A$ belonging to a trustworthy A to the node that is declaring it.

If the nodes are programmed to allow any size of group, there is nothing to stop a node controlled by the intruder joining into HCBK1. The result would be that the members of $\mathbf{G}$ have some *INFO* for nodes that are outside the group defined by the empirical channels. This is fine provided they do not assume that all the nodes that have participated in the run are in the intended group, or a check is performed after the run.

An alternative, which only makes sense if all the nodes in the group are assumed to be trustworthy, is for each node to check that the number of participants corresponds to the expected number. Since each one is present, it then follows no-one else is. We will call this protocol HCBK2.

If each $INFO_A$ contains a way of sending A data privately, say a public key (which needs not be certificated or long term) or a Diffie–Hellman token, then we could replace the broadcast Message 3 by some means of propagating k_I securely. This could take the form of a separate message from I to each S, or some tree of propagation amongst the S rooted at I. Upon successful completion of the protocol the group would then have a shared secret, namely k_I. Since it is vital that a shared secret is not shared with untrustworthy nodes, variants of this form are only useful on the assumptions that (a) all members of $\mathbf{G}$ are trustworthy and (b) that the number of participants is checked as in HCBK2. Clearly this represents a class of potential variant protocols, but we name them all under the heading of HCBK3.

This protocol works because, following its successful completion, we know (with likelihood $1 - 2^{-b}$) that k_I has only been sent to trustworthy participants, namely it really is a secret unknown outside the group $\mathbf{G}$.

Recall that these protocols depend crucially on the initiator I being trustworthy: a corrupt initiator could use a birthday attack essentially like the one we described earlier.

One situation where this is definitely not an issue is when the slave devices themselves have no need of security, as when the user of the initiator is seeking to connect his or her laptop to a number of wireless peripheral devices. That person must be sure that the connection is precisely to those devices that are trusted because of their context, labelling, etc. In that case there is no need for empirical channels *from* the initiator *to* the slave devices. All we require is that these devices can signal the initiator (probably via some display that the initiator's user can see) to convey Message 2b and the digest value from Message 4.

This would work for all three of the variants described above: HCBK1, 2 and 3. We will call the resulting, simplified protocols AHCBK1, 2 and 3, on the grounds that they are definitely *asymmetric*. (The original protocols

[8] For details of the attack, please see Appendix.

are neither properly symmetric, thanks to the role of the initiator, nor asymmetric, since their overall goals are symmetric.)

4. Symmetrised group protocol

The main protocol we present in this section was devised by the authors in February 2006.

The protocols in the previous section all rely crucially upon I being trustworthy: what are we to do if there is no node that is uniformly trusted or it is hard to select one, but we still want a *local PKI* which authenticates the $INFO_A$s of all trustworthy nodes? What we would like to achieve instead, is that a successful run of the protocol correctly authenticates all the trustworthy parties to each other irrespective of what the others may have done.

In order to do this we identify the following second principle, derived from the design of HCBK1:

P2 A node A is safe from effective manipulations of its final hash or digest d to equal others in a hash anomaly provided that there is a point at which the following things are both true.

(a) A is committed to its final value $d = digest(k^*, INFOS)$, though it may not yet know it.
(b) There is a value k_A which A knows, randomising the calculation of k^*, which (i) no other party can know and (ii) no other party can have used in the protocol in a way that has influenced A's final digest d.

Note that this is true of the initiator I and the value k_I in HCBK1 at the point where I has just sent Message 2a. In that protocol I is completely committed to its digest at the point $longhash(k_I)$ is revealed, so no other node can have used $longhash(k_I)$ in a message; the purpose of clause (ii) above will become apparent later when more than one node is responsible for the value of the digest key.

Before we can establish anything formally, we need to be precise about the idea used above that k_A randomises the calculation of k^*. We will assume that A calculates k^* by some formula from k_A and perhaps some values communicated to it by other nodes in the protocol—necessarily values it is committed to at the point discussed in P2. Thanks to assumption (b)(ii) we know all those values are independent of k_A. What we mean by "randomises k^*" is that if k_A varies uniformly and randomly across its range then k^* also varies uniformly and randomly across its own, for other values being fixed.

The most straightforward way of achieving randomisation is for A to calculate k^* as the XOR of the set of k_A's it wants to construct it from.

The fact that this definition is symmetric is an advantage in group protocols because it does not matter what order each node records the same group in. From here on we will assume that this XOR method is used, and in fact we have already taken account of this in the definition of a digest function: it lies behind the "$\oplus\ \theta$" in the second part.

In HCBK, **P2** applies to the initiator relative to k_I, which is the only contributor to the final digest key. That protocol relies on much more subtle reasoning in respect of the slave nodes, as shown by our reasoning in the previous section and the principle of Messages 2 and 4 being aligned that we had to adopt. If the slave nodes had been able to follow **P2**, there would have been no need for this.

Our second sort of protocol is designed so that all nodes can rely independently on **P2**. Therefore each node will now need some value made up specially for this purpose, which is fresh and unpredictable. Let us call this value k_A. The protocol is now:

Symmetrised HCBK protocol
1. $\forall A \longrightarrow_N \forall A' : A, INFO_A, longhash(A, k_A)$
2. $\forall A \longrightarrow_N \forall A' : k_A$
3. $\forall A \longrightarrow_E \forall A'$: users compare $digest(k^*, INFOS)$ where k^* is the XOR of all the k_A's for $A \in \mathbf{G}$

The following notes explain these messages.

- **Message 1**: introduces the information, $INFO_A$, each party A wants to authenticate and a long hash of its key k_A. The identity A is included in this longhash to ensure that the intruder posing as $C \neq A$ cannot simply copy A's key k_A by copying its longhash value to negate A's randomising effect on k^*, which leads to a reflection attack.[9] After this message each node should have all the information it requires about the other nodes except for the values k_A, and furthermore should be committed to each of these values in the sense that when told the k_A's it will be able to check each one.
  At the point when the sending and receiving of this message is complete, it follows that every node A is committed to some final digest value d_A, knows one of the antecedents (k_A) of this final digest that no-one else does. It is certain at this point that no party other than A can know the final digest—and very likely that A doesn't know it either. The distinction between being *committed* to a value and *knowing* it is immensely important. From this we know that **P2** applies.

- **Message boundary**: there has to be some moment at which a node decides it has finished inputting new Message 1's. This might be determined by some timeout, or some message sent from one of the nodes (empirically or over the general network). It is clearly in nodes' interest that they all make correct decisions on this, for otherwise they will not agree. One can imagine them attempting to synchronise by agreeing on a hash of the Message 1's they know about over the Dolev–Yao channel: that might well serve a useful purpose since it would guard against involving humans in empirical communication when there is no point.
  Whatever mechanism they choose does not matter provided it does not involve them revealing the keys k_X's to each other. For it is absolutely vital that none of them accepts any further Message 1 after its own key k_X is revealed.

- **Message 2**: each node broadcasts its unguessable key to all other nodes once it is committed to its final digest value. Having received all these keys, each node can check the correctness of all the long hashes received from Messages 1. If there is any thing unmatched regarding the long hash values, the node will abort and presumably tell the rest of the group that this has happened.
  Essentially these broadcasts expand the longhashes of the Messages 1 into something the nodes can understand.

- **Message 3**: has the members of **G** display and compare the value of digests through the empirical channel. Notice that, like both the previous protocols we have considered, this digest follows **P1** and includes the whole data of the protocol.

4.1. Protocol analysis

We shall call this the SHCBK protocol, for *Symmetrised Hash Commitment Before Knowledge*. The final result is that the members of **G** are authenticated to each other as the owner of the information they have introduced. The fact that SHCBK achieves its goal of authenticating the (A, $INFO_A$)s is a consequence of this following result.

Theorem 2. *Suppose a protocol calls for the agreement on the digest value $d = digest(k^*, INFOS)$. Suppose further that the trustworthy node A makes a contribution k_A towards its own calculation of k_A^* via XOR, and that each other trustworthy node B calculates its k_B^* as a similar XOR.*

Then if P2 applies to A and the value k_A^* then the likelihood that $digest(k_A^*, INFOS_A) = digest(k_B^*, INFOS_B)$, for a second trustworthy node B for which $(k_A^*, INFOS_A) \neq (k_B^*, INFOS_B)$ is smaller than or equal to 2^{-b}.

[9] We could build a check into our protocol by saying that no node A accepts its own value of $longhash(k_A)$ from another user, however, putting the name in makes it clearer. The same reflection attack was also reported in the papers of Čagalj et al. [8], and that is why their pairwise protocol concatenates a single bit (0 and 1) in front of each *INFO*. The same thing is done with the two-way authentication scheme of Vaudenay [37], but he did not make it clear why. Fortunately, the reflection attack does not work against HCBK as there is only a single cryptographic hash generated by the initiator, $longhash(k_I)$.

Proof. As we have argued above, whatever other (dishonest) nodes pick for their k_C's, these values cannot be related to either k_A^* or k_B^*, thanks to the use of XOR and the identities included in $longhash(A, k_A)$ and $longhash(B, k_B)$ in Messages 1 that avoid a reflexive attack.[10] As a result, the actual values of both k_A^* and k_B^* are uniform random variables whose values no node knew at the point where they agreed to finish inputting the first messages.

Clearly the intruder cannot prevent A having k_A in its XOR, nor can it prevent B from having k_B. It can, if it chooses prevent one or other of A and B having the other's key in the set it XORs. It is easy to see that unless it allows them each to have the other's key the values k_A^* and k_B^* are themselves independent uniform random variables as k_A and k_B vary. In this case, by the first part of the specification of a digest function, the digest values $digest(k_A^*, INFOS_A)$ and $digest(k_B^*, INFOS_B)$ are independent with the probability of 2^{-b} of being equal whether $INFOS_A$ and $INFOS_B$ are equal or not.

So we can concentrate on the case where each gets to see the other's key. In that case, there is no need for the intruder to allow A and B to see the same set of other k_C's. If all other nodes are under the control of the intruder, we will have $k_A^* = k_A \oplus k_B \oplus \phi$ and $k_B^* = k_A \oplus k_B \oplus \psi$ for values ϕ and ψ controlled by the intruder. So there will be a value $\theta = \phi \oplus \psi$, independent of k_A and k_B and possibly picked by the intruder such that $k_A^* = k_B^* \oplus \theta$ as these values vary randomly.

The probability that $digest(k_A^*, INFOS_A) = digest(k_B^*, INFOS_B)$ when $INFOS_A \neq INFOS_B$ is then smaller than or equal to 2^{-b} by the second part of the digest specification, presented in Section 3. $\square$

Note that this result, coupled with Theorem 1, establishes that B should be in a position to associate $(A, INFO_A)$, with the empirical identity that owns A, confidently, even if nodes other than the two of them are not trustworthy. Thus SHCBK does indeed authenticate the $INFO_A$'s of trustworthy nodes to each other even if corrupt nodes are in **G**.

It is important to note that our protocols do not supply evidence that nodes *are* trustworthy: mutual trust has to be brought into the protocol from outside, or possibly be established subsequent to the protocol run based on nodes' later communications.

Calling the basic protocol SHCBK1, it can be extended by a count of nodes to create SHCBK2 for the case where all nodes are assumed to be trustworthy. Asymmetric versions are also possible; they use more computational effort than the asymmetric versions of HCBK, but avoid the *commit* signals required there.

5. Digest functions

The specification of the digest function given in Section 3 has similarities to universal hash functions originally proposed by Carter and Wegman in [11,38] although our specification is more restrictive because of the presence of θ (the two definitions are the same if θ is fixed to 0).[11] However, there does not seem to have been much work on the *short-output* universal hash functions and trying to exploit short outputs to decrease the amount of computation compared to calculating long-output hashes. Most work on cryptographic hash functions concentrates on ones that are collision- and inversion-resistant: these properties are not required of our digest functions, which is just as well since the low number of output bits render them unachievable.

What we are going to do in this section is to give a very brief analysis of what has been done in the literature in constructing functions with similar purposes. We then move on to propose our own ideal digest framework, and develop ideas for efficiently implementing it in both hardware and software using pseudo-random number generation (PRNG).

[10] In fact, of course, the intruder can use any function derived from A's or B's own longhash, as a k_C, but since it has no way of relating these longhashes back to k_A and k_B, such values are no better than independent for cryptographic purposes.

[11] Our style of use of digest functions is very different from that of [38,36] since our keys vary dynamically and randomly at run time, whereas in the calculation of message authentication codes (MACs) they are fixed for all time. As has been demonstrated in the proof of Theorem 2, the way in which our keys are agreed between nodes at run time can be manipulated by an attacker in a way that means that different nodes' keys may be relatively shifted by a θ known to the attacker. The inclusion of the θ shift in our definition of a digest is to ensure that this type of activity can never benefit the intruder.

5.1. Background information

There are some important points we wish to make about the computation of the digest values. In order to make things simple and secure, the parties need to sort all of the pairs $\{A, INFO_A\}$ in alphabetic order say, before they concatenate all of them into *INFOS*. It is also normal to require that all the names A and public keys (if in $INFO_A$) are distinct.

Our second comment relates to the randomising effects of the digests. It would be a great mistake to compute $digest(k^*, INFOS)$ as some function of k^* and some similar length $digest'(INFOS)$, which it might be tempting to do. This is because an intruder could then—during the exchanges of Message 1—manipulate the sets $INFOS_A$ heard by the different nodes A provided they will all compute the same $digest'(INFOS_A)$. The fact that the intruder cannot predict at this stage what the final digests will be (not knowing k^*) would be irrelevant, since it would know they will all calculate the same value. As a result, what we need to compute is the keyed digest of *INFOS* with respect to key, k^*, with k^* fundamentally embedded in the calculation.

With this in mind, a number of different schemes for computing the digest values have been proposed by Pasini, Vaudenay, Gehrmann and others in [31,16,7]. Pasini, Vaudenay, Gehrmann in [31,16] as well as the Bluetooth white-paper [7] suggested the following scheme:

$$digest(k^*, INFOS) = trunc_b\left(hash(k^* \parallel INFOS)\right)$$

where *hash* is a cryptographic hash function such as SHA, TIGER or a block cipher such as DES. The $trunc_b()$ function truncates to the leading b bits. We make two observations about this. The first is that the definition of an inversion- and collision-free hash function does not normally specify the distribution of individual groups of bits: if $h(x)$ is such a function then so is $h'(x) = \langle 1\rangle^b \cdot h(x)$, which would clearly be useless in their protocol as $\forall x$ and $\forall k^*$ we have:

$$digest(k^*, x) = trunc(h'(k^* \parallel x)) = trunc(\langle 1\rangle^b \cdot h(k^* \parallel x)) = \langle 1\rangle^b$$

It follows that the standard specification of a hash function is useless in establishing that the above function comes close to our specification: a specific analysis would be required for any particular function proposed. The second observation is that computing a longhash that operates on long words is certainly expensive and does not exploit the short bit-length output of the digest function. We will discuss the relative complexity of hashing and digests in Section 6.

Taking a different approach, Gehrmann and Nyberg in [17] proposed using an error-correcting code such as the Reed-Solomon code to construct the digest function. This has the advantage of having a coherent mathematical structure but on the other hand the algorithm limits the bit-length of the input message to some fixed number such as 128 or 256. As a consequence, to digest any significant amount of data, the algorithm must firstly compress the input message into that number of bits by using a cryptographic hash which is inefficient as discussed above. This feature also makes the scheme be potentially vulnerable to attacks should the intruder find (off line) a collision on the cryptographic hash. The reason for this weakness is that the input message is not entirely linked to the key in computation as discussed in the second paragraph of this section.

In summary, both these approaches rely on applying a standard hash function to an object at least as large as *INFOS*, something which is much less efficient than necessary. Their disadvantage will grow as the size of the group (and hence the size of *INFOS*) increases, and this will be discussed in Section 6.

5.2. Matrix product construction of a digest function

Let us recall the formal specification of the digest function: as we vary the key k^*, we always have:

(1) $digest(k^*, M)$ is uniformly distributed for any fixed M.
(2) And for any fixed θ and $M \neq M'$:

$$\Pr(digest(k^*, M) = digest(k^* \oplus \theta, M')) \leqslant \frac{1}{2^b}$$

In order to satisfy the above specification, we need to have the probability of $digest(k^*, M)[i] = digest(k^* \oplus \theta, M')[i]$ be smaller than or equal to $\frac{1}{2}$ for $i = 1, \ldots, b$ when $M \neq M'$, and that the probabilities for different i are independent.[12] This means that a change in any non-zero number of bits of M must have a distinct random effect on every bit of the output.[13]

Suppose we want to construct a b-bit digest of a $(K-1)$-bit Message M. The first thing we do is to pad M with an extra 1-bit at the end, so its length becomes K with $M_k = 1$. We can build an *idealised* digest function as follows. Let us consider the following idealised framework: for $i = 1, \ldots, b$ and $j = 1, \ldots, K$, suppose that $R_{i,j}$ are independent uniform boolean-valued random variables (UBRV's) based on k^*.[14] Using matrix product, we define:

$$digest(k^*, M) = M \odot R$$

where the symbol $\odot$ represents the binary product of the vector M and the matrix R. This is, of course, the same as the b inner products of M and the columns of R. This is equivalent to defining

$$\forall i \in \{1, \ldots, b\} : d_i = \bigoplus_{j=1}^{K} (R_{i,j} \wedge M_j)$$

and

$$digest(k^*, M) = [d_1, \ldots, d_b]$$

where M_j is the jth bit of the input M.

Theorem 3. *The definition above satisfies the specification of a digest function provided that the $R_{i,j}$ are derived from k^* linearly (i.e., $R_{i,j}(k_1^* \oplus k_2^*) = R_{i,j}(k_1^*) \oplus R_{i,j}(k_2^*)$).*

Proof. This proof relies heavily on the standard fact that if U and V are any independent boolean random variable, and U is a UBRV, then $U \oplus V$ is also a UBRV.

The first use of this principle comes in the proof that $digest(k^*, M)$ is uniformly distributed. We know by construction that all the bits of this digest are independent (since the sets of the $R_{i,j}$ they are based on are disjoint). It follows that $digest(k^*, M)$ is uniformly distributed if all its bits d_i are. That in turn follows because $d_i = R_{i,K} \oplus X$, where the first term comes because we have assumed $M_K = 1$ and the second comes from the rest of the terms of the inner product creating d_i. Since $R_{i,K}$ is clearly independent of X and is UBRV, it follows that d_i is UBRV.

So suppose we have fixed $M \neq M'$ and θ. To prove the second part of the specification we need to show that the probability that $digest(k^*, M) = digest(k^* \oplus \theta, M')$ is 2^{-b}. The left- and right-hand sides of this equation are d_A and d_B, respectively, and let the $R_{i,j}$ derived from these two keys be, $R_{i,j}^A$ and $R_{i,j}^B$.

For all $i \in \{1, \ldots, b\}$, we then have:

$$d_i^A = \bigoplus_{j=1}^{K} (R_{i,j}^A \wedge M_j)$$

[12] $digest(k^*, M)[i]$ denotes the ith bit of the digest value $digest(k^*, M)$.

[13] This requirement is similar to the *strict avalanche criterion* [1], used in the design of S-Box of DES [39] and customised hash functions such as SHA or TIGER that guarantees any single or multiple changes in input bits have a random effect on every output bit.

[14] For them to be truly independent k^* would have to have far more bits than it actually does. That is why we call this the *idealised* digest. In the ideal model we will, for simplicity, identify the $R_{i,j}$ with distinct bits determined by i and j of the corresponding k^*. In practice we should aim to have them only subtly dependent, and in ways that are impossible to predict without knowledge of k^*. We will discuss this issue later.

$$d_i^B = \bigoplus_{j=1}^{K} (R_{i,j}^B \wedge M'_j)$$

We can see that the ith bit, t_i, of $digest(k_A^*, M) \oplus digest(k_B^*, M')$ is equal to:

$$\begin{aligned}
t_i &= \left(digest(k_A^*, M) \oplus digest(k_B^*, M')\right)[i] \\
&= d_i^A \oplus d_i^B \\
&= \left(\bigoplus_{j=1}^{K} (R_{i,j}^A \wedge M_j)\right) \oplus \left(\bigoplus_{j=1}^{K} (R_{i,j}^B \wedge M'_j)\right) \\
&= \bigoplus_{j=1}^{K} \left((R_{i,j}^A \wedge M_j) \oplus (R_{i,j}^B \wedge M'_j)\right) \\
&= \bigoplus_{\{j \in \{1..K\} | M_j \wedge M'_j\}} (R_{i,j}^A \oplus R_{i,j}^B) \\
&\quad \oplus \bigoplus_{\{j \in \{1..K\} | M_j \wedge \neg M'_j\}} R_{i,j}^A \\
&\quad \oplus \bigoplus_{\{j \in \{1..K\} | \neg M_j \wedge M'_j\}} R_{i,j}^B
\end{aligned}$$

It quickly follows from $k_A^* = k_B^* \oplus \theta$ and the linearity of the derivation of the $R_{i,j}$ that any failure of independence between $R_{i,j}^A$ and $R_{k,l}^B$ can only happen when $i = k$ and $j = l$.[15] This means that the three random variables from which t_i is formed above are independent. Our assumption that $M \neq M'$ means that the sets over which the last two are "summed" cannot both be empty. Whichever one is nonempty must (as the $\oplus$ of independent UBRVs) itself be a UBRV.

Since the t_i's are themselves independent in our ideal model, it follows that the probability that the two digests are equal is precisely smaller than or equal to 2^{-b}. $\square$

It was stated in [29,20,28] that the completely random matrix R could be replaced by a Toeplitz matrix—one with constant diagonal thanks to the relation $R_{i,j} = R_{i+1,j+1}$—where the same calculation was used to create a universal hash function, decreasing the required number of random bits from $K \times b$ to only $K + b - 1$. The same is true for digest functions and we will present the relatively complex proof of this in a subsequent paper.

This suggests that, in order to get a good digest, we need to get close to this ideal model with either a completely random matrix or a Toeplitz one. The most obvious way to do this is to use the k^* value to seed a suitable pseudo-random number generator (PRNG) as has been suggested by Krawczyk in [24,25] but otherwise follow the ideal model. Since the problem of deciding whether a typical stream is truly random or pseudo random has been well studied, for example there are a number tests for randomness that must be satisfied by any standard pseudo random number generation [19,26,18], it follows that if we use a good PRNG (one known to satisfy these tests) with a seed which is of the size of a typical cryptographic random number (say 160 or 200 bits), then for cryptographic purposes it should be essentially as good as the ideal model. In the following sections we will see two approaches to calculating the digest: the first is a hardware implementation of exactly this idea; the second

[15] In practice, if one wants to use a linear pseudo-random bit generator to derive $R_{i,j}$, then there will exist linear relations between any bit and some (says r) of its preceding bits in the random output stream, for example: $b_{n+r} = f(b_n \ldots b_{n+r-1})$, where $f()$ is a linear function. However, it is possible to make these relations highly unpredictable if we make the linear function/structure of the random number generator depend on the value of the key or part of it which is unknown to the intruder and every one else at the point when *INFOS* is committed. So what we might expect is as follows: $b_{n+r} = F_x(b_n \ldots b_{n+r-1})$, here x is derived directly from the key k^*. And the effect of this as well as the detailed construction will be analysed more extensively in a subsequent paper.

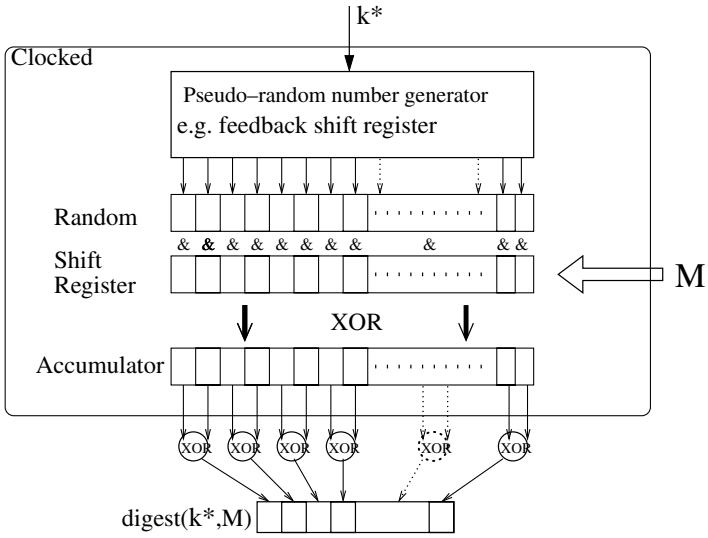

Fig. 2. Hardware implementation of a digest function.

is an approximation to this using operations that can be executed efficiently using standard microprocessor operations.

5.3. Hardware implementation

Suppose we are given input data *INFOS* with bit-length K. Suppose $r = m * b$ is some multiple of the desired output length b. We first need to generate $(K/m) + b$ pseudo-random numbers each of length r, seeded by k^*, in a register R on designated clock cycles. One possibility, illustrated in Fig. 2, is to use one or more linear feedback shift registers (LFSR) discussed in [24,25,19] to produce pseudo-random numbers, each seeded with the whole of k^* or some linear function of it. We then initialise a shift register S, and an accumulator register A, both with length r to standard values, possibly 0. The random number generators are designed to produce r bits on each cycle in a register R.

On each designated clock cycle, the register S is shifted by m bits, introducing the next m bits of *INFOS* or some standard values, possibly 0, if *INFOS* is already complete. By this means the whole of *INFOS* will have passed entirely though S after $(K/m) + b$ cycles. Also on each designated clock cycle, we enable the replacement of the accumulator register A by the previous value bitwise-exclusive-or-ed with the bitwise-and of the registers R and S.

$$A = A \oplus (R \wedge S)$$

After the $(K/m) + b$ cycles are completed, we partition the register A into b groups of m consecutive bits. The bits of each of the said b groups are then exclusive-or-ed to a single bit. Finally, the juxtaposition of the said b bits will form the digest value. That way each input bit has contributed once to each bit of the digest as in the software implementation. In addition, the only operators we use here are *Shifting*, *XOR* and *AND*, so this makes implementation done very fast in hardware.

The efficiency of the implementation can be improved if we switch to using a Toeplitz matrix of random values rather than the completely random one discussed above, since this dramatically reduces the number of pseudo-random bits required to be generated in the scheme.

If, as suggested above, the random bits are generated by LFSRs, then the outputs generated are linear functions of k^*. This means that this type of pseudo-random number generator, as well being very well understood and well behaved (see [19], for example), has the linearity property used in Theorem 3. We intend to discuss further ideas for the creation of PRNGs in a sequel to this paper.

5.4. Software implementation

The above calculation can, of course, be run in software as well as hardware. However, it would take many clock cycles without the implementation of special operations. In this section we show that a good approximation can be calculated using standard integer multiplication for half or whole word blocks implemented in most microprocessors rather than a customised hardware. Let us divide *INFOS* into b-bit blocks $\left[m_1, \ldots, m_{t=\frac{K}{b}}\right]$. We then generate pseudo random b-bit blocks r_i based on k^*.[16] If we define

$$S = \bigoplus_{i=1}^{t} (m_i \times r_i)$$

The integer multiplications $m_i \times r_i$ lead S to have $2b$ bits. We finally set

$$digest(k^*, INFOS) = S_1 \oplus S_2$$

where S_1 and S_2 are two halves of S.

We note that multiplication together with XORing let every bit of the input influence every bit of the output very much uniformly. Unfortunately, there may be some asymmetry due to the carry bits in multiplication. Furthermore, in software implementation, only b random bits of r_i influence how each bit of m_i maps into the 16 bits of the digest output while the ideal model suggests there should be b^2. In other word, the analogues of the R_{ij} from the ideal model we are using cannot be completely independent.

To increase this independence we could additionally add one or more terms of the form $r_i'' * (r_i' \wedge m_i)$, where r_i' and r_i'' are different series of pseudo-random numbers, to our accumulator. Increasing the amount of calculation like this could move us closer to our idealised model, and the number of terms of the form $r_i'' * (r_i' \wedge m_i)$ used represents a trade-off between the efficiency and quality of the digest. Further work is required to decide if this is worthwhile.[17]

6. Efficiency

It seems reasonable to measure the efficiency of protocols in this class in two ways: the amount of empirical, or human, effort required to complete them; and the amount of processing required at the nodes.

6.1. Empirical work

The major item of work for the humans is probably the sending and receipt of the final digest value, and the effort required to check equality. In the case where a user can broadcast empirically to all other nodes (as with a set of people in a room), the most efficient way of performing this check is for one person to announce his/her value d_0 and the rest to check that their values all equal d_0. Depending on circumstances they might then each have to announce definite equality, or only announce inequality.

It seems clear, as argued in [37,34], that it is impossible to bound the intruder's chance of success to 2^{-b} by comparing (explicitly or implicitly) less than b bits of information. Given pre-knowledge of the size of the group, the size check in protocols labelled 2 and 3 is essentially free; it seems impossible to account for the difficulty of performing it in other circumstances (though in the protocols with an initiator it simply means that the number of Message 2b's received by I corresponds with the number of nodes' *INFO*s that are digested).

It therefore seems that all our protocols are essentially optimal in the amount of security they provide for a given amount of human effort, *except* that in the non-symmetrised cases (HCBK) there is the work involved in the sending and receipt of Message 2b, which is a constant and certainly less than the effort required for Message 4. We will see shortly that this represents one side of an interesting trade-off.

[16] Any high quality pseudo-random generator could be used here. Similar to the hardware implementation, we can use a linear feedback shift register seeded with k^*, or several seeded with parts of k^*, this can be implemented extremely fast in either software or hardware.

[17] We have recently discovered that this software model can, with a small modification, satisfy the digest specification to the same extent as the Toeplitz adaptation of the ideal model. In this way only $K + b - 1$ (pseudo) random bits are required.

6.2. Assumptions about processing cost

Let W and B be the number of words and, respectively, bits required to hold a long hash value: in [37], Vaudenay suggests perhaps B =160 bits, so we assume $W = 5$. He also suggests that 15 or 16 bits are reasonable choices for b, the width of the digest which is 1 word in this case, and we will adopt that too. We also assume that nonces and other strong cryptographic values have the same length B. Aside from the protocols labelled 3, the only processing effort required in implementing our protocols is the computation of long hashes and digests. In order to assess the complexity of our protocols we have to have a model of the complexity of computing hashes and digests. It is clear that the cost of computing the b-bit output $hash_b(INFOS)$ increases linearly with the length of $INFOS$. It also seems clear that it will increase significantly with b, and a simple model in which each word of a running temporary value of length b is combined with each input word suggests our overall model might be $b \times length(INFOS)$, as indeed does the idealised model presented in the previous section. Therefore we will adopt that assumption in the following analysis. Since well-known hash algorithms tend to be fixed width, and vary significantly in their individual costs, it is hard to be too definite about this rule. Our analysis of SHA-1 shows it to have a cost perhaps 5 times that of the digest algorithm we described above, based on the random number generator quoted in the footnote earlier.

6.3. Processing cost of HCBK and SHCBK

It follows that the total processing cost of the non-symmetrised protocols labelled less than 3, with a group of size N and where the word-length of all the $INFO$s is M, at every node is

$$W \times W + M = 25 + M$$

This results from one longhash (both the input and output lengths are of W in words) and one digest (the input length is M words and the b-bit output is 1 word) computed by each party.

In the symmetrised case there is more work to do since now each node has to check $N - 1$ long hashes and create one.[18] Therefore the above quantity increases to

$$N \times W \times W + M = 25N + M$$

This, of course, is the other side of the trade-off mentioned above.

6.4. Comparison with the processing cost of Vaudenay's and Čagalj's schemes

Vaudenay's protocol [37], in its basic form, relates only to the transmission of a message from one party to another. In order to compare it with ours we need either to restrict our protocols to this function or to expand Vaudenay's so that it achieves the broadcast of a message from each member of a group to each other. We can do both of these things.

We will assume that the commit scheme, used in [37] to commit a message of length M and a nonce R_A of length b bits = 1 word, takes $max(M, W)$ words as input. Since the security of the scheme is equivalent to a cryptographic hash, it seems to require randomisation that introduces additional nondeterminism to that introduced by R_A. This is equivalent to adding a hidden variable of length $W - b$. We will assume, for ease of calculation, that $M \geqslant W$.

With $W = 5$ it follows that for transmission of a single message of size M this protocol requires, at each of the two nodes,[19] processing of order

[18] In both our symmetrised scheme and the two-way authentication protocol of Vaudenay [37], the inputs of the longhash and the commit scheme include an identity or a single bit to avoid a reflection attack. However, as both of them are very short, we ignore them in our analysis.

[19] It might be clearer to point out that W is also the word length of both the commitment c, and the decommitment d. In Vaudenay's scheme A has to compute the function $commit()$, whereas B computes $open()$. Both of the functions are equivalent in term of computation cost.

$$M \times W = M \times 5$$

Our symmetrised protocol does this in $25 + M$.

We observe that both Vaudenay's and Čagalj's protocols can be extended to a group protocol that achieves the same goal as our schemes: each node has to commit once and open (or decommit) $N - 1$ times, and no digest is required. (The users will finally compare the *XOR* of one short random string per node.) If M is the total size of all the *INFOs* in our protocols, then the equivalent message that each party in Vaudenay's or Čagalj's group version commits to will be of length $\frac{M}{N}$. In order for the commit scheme to have an equal level of security as our long hash, the lengths of both the random data of the input of the commit scheme and its output need to be W as discussed above. As a result, the processing cost of each party in Vaudenay's group version is approximately

$$N \times W \times \left(\frac{M}{N}\right) = M \times W = 5 \times M$$

As M is the concatenation of N pieces of public information $INFO_A$, we have:

$$M = N \times wordLength(INFO_A)$$

So the difference between Vaudenay's group protocol and SHCBK is:

$$\begin{aligned} 5M - (25N + M) &= 4 \times M - 25 \times N \\ &= 4 \times N \times wordLength(INFO_A) - 25 \times N \\ &= 4 \times N \times (wordLength(INFO_A) - 6.25) \end{aligned}$$

As the word length of $INFO_A$ will be always much longer[20] than 6.25 words that is equivalent to only 200 bits, this will normally be significantly more expensive than our protocols.

It seems clear that our protocols are the more efficient in terms of computational power because we followed **P1**: we have only had to bind the messages cryptographically to the level required for human interaction. Both Vaudenay and Čagalj chose to bind the messages to random values earlier, which would have been subject to a combinatorial attack had they not done so with more complex cryptography. The probability of a successful attack on either their protocols or ours is essentially 2^{-b}.

7. Conclusions and future work

In this paper, we have analysed the strengths and weaknesses of a number of protocols that form a secure network using empirical channels. We have introduced two new classes of such protocol: one (HCBK) that relies on a trustworthy initiator, and one (SHCBK) that allows arbitrary group members to be corrupt. The efficiency of these two classes (aside from the "committed" signals for HCBK) is as good as the best from other, independently discovered protocols, in terms of human effort. There is every reason to believe that this is optimal, since 2^{-b} from b bits communicated is exactly what one would be confident of obtaining against any protocol using a man-in-the-middle attack.

We have shown how the principles **P1** and **P2** lead to the design of correct protocols, in which we avoid combinatorial attacks creating digest anomalies by distinguishing carefully between when a node is committed to a value and when it knows it.

We have shown how data can be digested much faster than it can be hashed, and begun to develop a satisfying theory of digest functions as well as building some possible implementations of them that rely on the properties of pseudo-random numbers. In a subsequent paper we will analyse the properties of digest functions in much more depth, including giving the proof of the applicability of Toeplitz matrices in the ideal model.

We have briefly introduced the concept of a *local PKI*, that is in effect the result of the run of one of our protocols, since they bind information such as public keys, identities and context together in an authenticated way. It is natural to ask how one can extend this analogy to allow for adding nodes, forming the union of

[20] For example: $INFO_A$ should at least contains a public key of A, which is 1024 bits or 32 words already.

two such groups etc. This of course raises interesting questions of how trust based on confidence in particular (initiator) nodes or perhaps subgroups of **G** can extend in transitive ways. This will be a topic for future research.

It is natural to ask how protocols of this form fit into the standard models and analysis tools for cryptographic protocols. The answer is that our protocols are rather outside the standard models for two orthogonal reasons. The first is that they are *group* protocols with an arbitrary number of participants: most methods are only fully developed for protocols with a small fixed number.

The second is that they are intended to counter a much stronger attacker model than exists in the standard models: one who can perform combinatorial searches. We are developing a modified version of the standard CSP model for protocols that incorporates such a strong attacker and expect to report on that in a subsequent paper.

The availability of protocols such as these immediately suggests a wide range of potential applications across a wide range of domains. We see future research and development in this area as important. One topic that will be important here is that of how humans can, efficiently and reliably, compare digests. There is likely to be a tension between ease of use and ensuring compliance with the protocol. One way to compare two digests is to inspect them visually and press "OK" or "ABORT" as appropriate. This is easy but humans can easily press "OK" without actually checking. On the other hand, one machine in the group may display the digest, which has to be typed into all the others, which check if it corresponds to their own values. This is more work but cannot be circumvented. Therefore human factors work will be important if this type of protocol is to be used widely.

Acknowledgments

Long Nguyen's work on this paper was supported by studentships from QinetiQ Trusted Information Management and the Ministry of Education and Training of Vietnam. Roscoe's was partially supported by funding from the US Office of Naval Research.

We thank Michael Goldsmith, Sadie Creese and Irfan "Zak" Zakiuddin for their influence on the development of earlier ideas on this subject, and continuing discussions. It was Zak who originally pushed the area of bootstrapping from minimal assumptions. We thank Richard Brent for helping us understand pseudo-random numbers.

We are grateful to anonymous referees whose detailed comments allowed us to greatly improve the paper.

Appendix

A: Further assumption required for HCBK

In order to make the HCBK protocol secure we require that the non-initiator nodes are never allowed to get involved in other runs between the acknowledgement signal sent in Message 2b and the final agreement on the short digest value. Otherwise the protocol will be vulnerable to a subtle attack, discovered by Roscoe in [34], that again makes use of combinatorial search if it is possible for the intruder to block the empirical channel.

We shall consider the situation where there are two nodes, the initiator A and the non-initiator B. So in the first run α, the protocol looks like this:

$$
\begin{array}{llll}
0.\alpha. & A & \longrightarrow_N B & : A \\
1.\alpha. & A & \longrightarrow_N B & : A, INFO_A \\
 & B & \longrightarrow_N I(A) & : B, INFO_B \\
 & I(B) & \longrightarrow_N A & : B, INFO'_B \\
2.a.\alpha. & A & \longrightarrow_N B & : longhash(k_A) \\
2.b.\alpha. & B & \longrightarrow_E A & : \text{B's acknowledgement} \\
3.\alpha. & A & \longrightarrow_N I(B) & : k_A \\
 & I(A) & \longrightarrow_N B & : k'_A
\end{array}
$$

As can be seen from the run α, in Messages 1 and 3, the original information $INFO_B$ and k_A have been replaced by the fake $INFO'_B$ and k'_A. This leads the protocol to three negative consequences:

- When B receives the incorrect key k'_A in Message 3, B cannot verify the correctness of the longhash sent in Message 2. So B rejects it and aborts the run.
- In the mean time, the intruder obtains the original key, and therefore can determine the final value d_α of the digest in this run.
- Also note that the initiator, A, does not have any idea about the status of the current run, so s/he just keeps waiting for Message 4a from B.

Meanwhile, the intruder starts a second run with B, posing as A:

$$\begin{array}{lll} 0.\beta.\ I(A) & \longrightarrow_N B & : A \\ 1.\beta.\ I(A) & \longrightarrow_N B & : A, INFO_A \\ \quad B & \longrightarrow_N I(A) & : B, INFO_B \end{array}$$

At this point the intruder knows all the information that B will use in run β to compute the digest, except that run's key k''_A. It can therefore search for an k''_A that will digest, together with that information, to d_α. If it succeeds it continues:

$$\begin{array}{ll} 2.a.\beta.\ I(A) \longrightarrow_N B & : longhash(k''_A) \\ 2.b.\beta.\ B \longrightarrow_E A & : \text{The signal will be blocked by the intruder.} \\ 3.\beta.\ \ I(A) \longrightarrow_N B & : k''_A \end{array}$$

One of the main features in this run is that the acknowledgement signal sent over the empirical channel from B to A in Message 2.b.β is blocked by the intruder. This avoids the possibility that the trustworthy initiator might spot this message and realise that an attack is taking place.

After receiving k''_A from Message 3.β, B will be able to check the correctness of the longhash sent in this run. The flaw of the protocol becomes apparent when B, thinking she is in the run β, displays and compares the digest of the run β with the digest of A in the run α over the empirical channel.

$$\begin{array}{l} 4.a.\beta.\ B \longrightarrow_E A : digest(k''_A, \{INFO_A, INFO_B\}) \\ 4.a.\alpha.\ A \longrightarrow_E B : digest(k_A, \{INFO_A, INFO'_B\}) \end{array}$$

So in the end, A and B agree the equality of two digest values that have different antecedents.

References

[1] See: http://en.wikipedia.org/wiki/Avalanche_effect.

[2] D. Balfanz, D. Smetters, P. Stewart, H. Wong, Talking to strangers: authentication in ad hoc wireless networks, in: Symposium on Network and Distributed Systems Security, San Diego, California, USA, 2002.

[3] M. Bellare, P. Rogaway, Entity authentication and key distribution, in: Advances in Cryptology—Crypto 1993, LNCS, vol. 773, Springer-Verlag, 1993, pp. 232–249.

[4] M. Bellare, R. Canetti, H. Krawczyk, A modular approach to the design and analysis of authentication and key exchange protocols, in: 30th STOC, 1998.

[5] M. Bellare, D. Pointcheval, P. Rogaway, Authenticated key exchange secure against dictionary attacks, in: Advances in Cryptology—Eurocrypt 2000, LNCS, vol. 1807, Springer-Verlag, 2000, pp. 139–155.

[6] See: www.bluetooth.com/developer/specification/specification.asp.

[7] Simple Pairing White Paper. See: urlwww.bluetooth.com/NR/rdonlyres/0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/_ urlSimplePairingWP_V10r00.pdf.

[8] M. Čagalj, S. Čapkun, J. Hubaux, Key agreement in peer-to-peer wireless networks, in: Proceedings of the IEEE, Special Issue on Security and Cryptography, vol. 94, no. 2, February 2006.

[9] R. Canetti, H. Krawczyk, Analysis for key-exchange protocols and their use for building secure channels, in: Advances in Cryptology—Eurocrypt 2001, LNCS, vol. 2045, Springer-Verlag, 2001, pp. 453–474.

[10] R. Canetti, S. Halevi, J. Katz, Y. Lindell, P. MacKenzie, Universally composable password-based key exchange, in: Advances in Cryptology—Eurocrypt 2005, LNCS, vol. 3494, Springer-Verlag, 2005, pp. 404–421.
[11] J.L. Carter, M.N. Wegman, Universal classes of hash functions, Journal of Computer and System Sciences 18 (1979) 143–154.
[12] S.J. Creese, M.H. Goldsmith, R. Harrison, A.W. Roscoe, P. Whittaker, I. Zakiuddin, Exploiting empirical engagement in authentication protocol design, in: D. Hutter, M. Ullmann (Eds.), Proceedings of 2nd International Conference on Security in Pervasive Computing (*SPC*′05), LNSC, vol. 3450, Springer, Boppard, Germany, 2005.
[13] S.J. Creese, M.H. Goldsmith, A.W. Roscoe, M. Xiao, Bootstrapping multi-party ad-hoc security, in: Proceedings of IEEE Security Track, 2006.
[14] S.J. Creese, M.H. Goldsmith, A.W. Roscoe, I. Zakiuddin, The attacker in ubiquitous computing environments: formalising the threat model, in: T. Dimitrakos, F. Martinelli (Eds.), Workshop on Formal Aspects in Security and Trust, Pisa, Italy, September 2003, IIT-CNR Technical Report.
[15] S.J. Creese, M.H. Goldsmith, A.W. Roscoe, I. Zakiuddin, Security properties and mechanisms in human-centric computing, in: P. Robinson, H, Vogt, W. Wagealla (Eds.), Privacy, Security and Trust within the Context of Pervasive Computing, Kluwer International Series in Engineering and Computer Science, Springer, 2004. Proceedings of Workshop on Security and Privacy in Pervasive Computing, Wien, April 2004.
[16] C. Gehrmann, C. Mitchell, K. Nyberg, Manual authentication for wireless devices, RSA Cryptobytes 7 (1) (2004) 29–37.
[17] C. Gehrmann, K. Nyberg, Security in personal area networks, in: C.J. Mitchell (Ed.), Security for Mobility, IEE, London, 2004, pp. 191–230.
[18] O. Goldreich, L.A. Levin, A hard-core predicate for all one-way functions, in: Annual ACM Symposium on Theory of Computing, 1989, pp. 25–32.
[19] S.W. Golomb, Shift Register Sequences, Aegean Park Press, 1981, ISBN: 0894120484.
[20] J. Hastad, R. Impagliazzo, L.A. Levin, M. Luby, A pseudorandom generator from any one-way function, SIAM Journal of Computer (1999).
[21] J.-H. Hoepman, Ephemeral pairing on anonymous networks, in: D. Hutter, M. Ullmann (Eds.), 2nd International Conference on Security in Pervasive Computing (*SPC*′05), LNSC, vol. 3450, Springer, Boppard, Germany, 2005, pp. 101–116.
[22] J.-H. Hoepman, Ephemeral pairing problem, in: 8th International Conference Fin. Crypt., LNCS, vol. 3110, Springer, 2004, pp. 212–226.
[23] M. Jakobsson, S. Wetzel, Security weaknesses in bluetooth, in: CT-RSA 2001, LNCS, vol. 2020, Springer-Verlag, 2001, pp. 176–191.
[24] H. Krawczyk, LFSR-based hashing and authentication, in: CRYPTO 1994, LNCS, vol. 839, 1994, pp. 129–139.
[25] H. Krawczyk, New hash functions for message authentication, in: EUROCRYPT 1995, LNCS, vol. 921, 1995, pp. 301–310.
[26] D. Knuth, The art of computer programming, Seminumerical Algorithms, third ed., vol. 2, Addison-Wesley, Reading, MA, 1997, ISBN: 0-201-89684-2.
[27] G. Lowe, Breaking and fixing the Needham–Schroeder public-key protocol using FDR, in: Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol. 1055, Springer-Verlag, 1996, pp. 147–166.
[28] M. Luby, Princeton Computer Science Notes, Princeton University Press, 1996.
[29] Y. Mansour, N. Nisan, P. Tiwari, The computational complexity of universal hashing, in: Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 1990, pp. 235–243.
[30] L.H. Nguyen, A.W. Roscoe, Efficient group authentication protocol based on human interaction, in: Proceedings of Workshop on Foundation of Computer Security and Automated Reasoning Protocol Security Analysis, August 2006, pp. 9–31.
[31] S. Pasini, S. Vaudenay, SAS-based authenticated key agreement, in: Public Key Cryptography—PKC'06: The 9th International Workshop on Theory and Practice in Public Key Cryptography, LNCS, vol. 3958, Springer, 2006, pp. 395–409.
[32] S. Pasini, S. Vaudenay, An optimal non-interactive message authentication protocol, in: Topics in Cryptology—CT-RSA'06: The Cryptographers' Track at the RSA Conference 2006, LNCS, vol. 3860, Springer, 2006, pp. 280–294.
[33] T. Peyrin, S. Vaudenay, The pairing problem with user interaction, in: SEC 2005, 2005, pp. 251–266.
[34] A.W. Roscoe, Human-centred Computer Security. See: http://web.comlab.ox.ac.uk/oucl/work/bill.roscoe/publications/113.pdf, 2005.
[35] F. Stajano, R. Anderson, The resurrecting duckling: security issues for ad-hoc wireless networks, in: Security Protocols 1999, LNCS, vol. 1976, Springer-Verlag, 1999, pp. 172–194.
[36] D.R. Stinson, Universal hashing and authentication codes, in: Advances in Cryptology—Crypto 1991, LNCS, vol. 576, Springer-Verlag, 1992, pp. 74–85.
[37] S. Vaudenay, Secure communications over insecure channels based on short authenticated strings, in: Advances in Cryptology—Crypto 2005, LNCS, vol. 3621, Springer-Verlag, 2005, pp. 309–326.
[38] M.N. Wegman, J.L. Carter, New hash functions and their use in authentication and set equality, Journal of Computer and System Sciences 22 (1981) 265–279.
[39] A.F. Webster, S.E. Tavares, On the design of S-boxes, in: CRYPTO 1985, LNCS, vol. 218, Springer Verlag, 1986, pp. 523–534.
[40] F.-L. Wong, F. Stajano, Multi-channel protocols, in: Proceedings of the 13th International Workshop on Security Protocols, LNCS, Cambridge, England, 2005.