Model checking cryptographic protocols subject to combinatorial attack

A.W. Roscoe, Toby Smyth and Long Nguyen
Oxford University Department of Computer Science
Parks Road, OX1 3QD, Oxford
Email: {Bill.Roscoe,Toby.Smyth,Long.Nguyen}@cs.ox.ac.uk

August 25, 2011

Abstract

We introduce an approach to model checking cryptographic protocols that use hashing too weak to resist combinatorial attacks. Typically such hashing is used when an extremely low bandwidth channel, such as a human user, is employed to transmit its output. This leads to two opportunities for attack: deducing a weak value from its properties and discovering alternative ways to produce a given weak value. The first of these proves a natural extension to established protocol modelling approaches, but for the second we require something more novel. We propose an approach based on taking snapshots of the intruder memory.

1 Introduction

Over the past two decades, many sophisticated methods [5, 8, 11, 6, 22] and tools, for example [9, 4] have been introduced to analyse cryptographic protocols. Indeed it is now possible to input a proposed protocol into any one of a number of tools and get either the reassurance that it is secure or an attack that demonstrates it is not.

Such tools, however, rely on underlying assumptions about the cryptographic primitives that they use. Essentially all of them use the assumption of *perfect cryptography*, with some being adjustable to allow for particular algebraic and other weaknesses of particular forms of cryptography such as RSA and Vernam (bit-wise exclusive-or) encryption.

In particular they generally rely on the assumption that all cryptographic objects have sufficient entropy that there is essentially no chance of guessing one with a particular behaviour. An exception to this is [10], where Lowe showed how the CSP model of protocol analysis can be extended by giving the intruder the ability to *guess* certain sorts of value *that have already been introduced by other parties*, the main intended application being password guessing attacks. In that paper, Lowe studied the case that there might be a *verifier*: a value whose calculation in different ways enables an attacker to verify a guess.

In this paper we examine the class of protocols which attempt to achieve authentication by communication and comparison of short strings, and specifically consider how to modify the CSP models of protocols and intruders described in [18, 21] to accommodate combinatorial search. An extensive survey of such protocols can be found in [12]. We will see a number of these protocols later in this paper. All that a traditional protocol verifier could achieve with one of these protocols is one or other of finding an attack that does not depend on the weakness of any value, or proving that no attack like this exists.

The short strings that are communicated over the low-bandwidth or human channel can come into existence in these protocols in different ways. They might come into existence as short strings randomly invented by one of the participants, meaning that they are naturally regarded as symbolic constants in an abstract protocol model. We will consider this class of protocol in Section 4 by extending the symbolic intruder model in a way similar to that described by Lowe in [10]. Alternatively, the short string might come into existence (typically in the hands of multiple protocol participants separately) through the application of a cryptographic primitive – a short hash or keyed digest function – that takes one or more (typically long and un-guessable) values and outputs a short one.

In Section 5, to deal with this type of protocol, we imagine that some of the hashing constructs used in our protocol may be sufficiently weak that the intruder has a realistic prospect of searching for a hash collision by some sort of trial and error involving the introduction of a large number of different fresh values. This requires a completely different approach in modelling. In particular we have to deal with the following problems:

- The symbolic representations used in protocol analysers for the protocol data yield no natural way of creating such collisions.
- Model checkers etc certainly cannot cope with anything like the range of fresh values that would, in practice, be used in a combinatorial search.

The main motivation for this work is provided by the class of protocols in which a human is used to transfer a value between two systems and thereby secure a previously insecure electronic connection. To make such protocols attractive – and in many circumstances to make them practical at all – this value needs to be much shorter than a strong cryptographic hash. Yet in a number of such protocols the value is either the short hash or a short keyed hash, or *digest* of information passed. If well designed, such use need not be insecure, because the protocols can prevent the attacker from performing any useful combinatorial search designed to create a hash collision.

We introduce a technique where the usual CSP model of cryptoprotocols is extended so that it behaves as though hashes containing data introduced freshly by the intruder can, in appropriate circumstances, behave as though they collide.

We discuss the application of our new methods to the protocols from Section 2, and then to protocols designed to bootstrap groups of agents. We then discuss prospects for various generalisations of our work, including the development of finite-state checks capable of proving arbitrary – rather than fixed finite – implementations of protocols.

While it is likely that the techniques developed in this paper could be employed in conjunction with other approaches to verifying protocols, the one we work with is modelling in the CSP process algebra [20] and running on the FDR refinement checker [18]. A number of CSP files that illustrate the techniques we discuss can be downloaded from [2] so that the reader can run them and adapt them for other protocols.

2 Non-standard authentication protocols

In the usual (Dolev-Yao) model of insecure communication networks, it is impossible to bootstrap a secure connection between two or more parties without something that allows them to verify communications from one another. In the traditional model of computer security, the parties are assumed to have a pre-existing cryptographic infrastructure for this: typically a network of trusted third parties or a PKI. There are a huge number of protocols for establishing an authenticated parties in cases like this.

However, as described in [12], in many circumstances relating to modern, frequently lightweight, mobile systems, such an infrastructure is either impractical or the name-based form of identity it is inappropriate: someone might well want to create an authenticated connection to a system they can identify clearly by context, but not by name.

Instead we assume that this context yields a separate, generally human-mediated and very low bandwidth, channel that the person or people wanting to build a secure network know provides a reliable and unspoofable link between the systems. Thus the intruder cannot successfully impersonate trustworthy parties on such *out of band* or *empirical* channels, though we generally do not assume that such communications are secret. In our discussion, such an empirical channel is denoted by $\longrightarrow_E$.

It is easy to see that we can in principle secure any existing channel that exists between our parties using a channel of this type: it is sufficient for Alice, in trying to create an authenticated and secret communication link with Bob, swaps public keys with him over an empirical channel that links them. Each then knows that the respective keys are bound to the parties they want to communicate with and can therefore use them to sign and send information such as session keys as though the public keys had been certified by some PKI.

A public key is, however, much too large a piece of data for the typical human to copy it willingly and accurately. As pointed out by Balfanz et al. [3] we could replace the public key by its hash using some standard cryptographic strength algorithm, and use this hash to check on the value of the public key send over an insecure but very high-bandwidth channel, including the Internet or WiFi, which is denoted by $\longrightarrow_N$. On the assumption that this hash is smaller than the key itself this requires less, though still far too much, human effort, but gives security that is just as good because we assume that it is impossible for the attacker to discover any hash collision against such a function. Thus the hash from a known origin provides a "certificate" that the key received over the insecure network comes from that origin also.

Balfanz et al. non-interactive protocol, [3]
1. $A \longrightarrow_N B : A, INFO_A$
2. $A \longrightarrow_E B : hash(A, INFO_A)$

If, as a concession to the human user who has to implement the empirical channel, we were to use a weak hash for the above, say with 20-30 bits, then an attacker could search through a large number of messages with the same format as the one that Alice (say) sends – it might well have done this in advance. If successful then it could replace Alice's version by its own, and Bob's check of the empirically sent hash would tally.

The work required by the intruder might reduce if the protocol were redesigned so that Alice and Bob send each other public keys pk_A and pk_B over an insecure channel and, rather than transmit two hashes empirically, they both compute $hash(pk_A \| pk_B)$ and compare these. For the task for the intruder now becomes to find pk'_A and pk'_B distinct from pk_A and pk_B such that $hash(pk_A \| pk'_B) = hash(pk'_A \| pk_B)$. He can therefore employ the *birthday attack* which means that he only has to compute approximately the square root of the total number of hashes to expect a collision. But in either case the opportunity to carry out combinatorial search means that a determined attacker could either eliminate or severely degrade the security of the protocol.

The protocols we are considering in this paper are intended to prevent such search. We give a number of examples in this paper: many more can be found in [12]. The following protocols, to help direct comparison, all attempt to transfer a piece of information, $INFO_A$, from Alice to Bob. In some cases they are simplified versions of protocols for binary or group mutual authentication. We will consider some group protocols and their verification in Section 7.

Our first example is a protocol introduced by Vaudenay [23], that makes use of an abstract *commitment scheme* that can be implemented using (strong) cryptographic hashing.

A probabilistic commitment scheme can be simply explained as follows: given data x, someone generating commitment values will build a pair (c, d), where c is the *commitment value* and d the *decommitment* value. These values are nondeterministic, i.e. they vary even when the same data x is inputted more than once, because the algorithm is seeded by some strong random value such as a 256-bit nonce that is generated on the fly. Given values c and d, any party can deterministically deduce x and check that the two were generated together. A commitment scheme must satisfy the following two properties:

- Hiding: given value c, it must be inconceivable that any party could deduce any information about x; and
- Binding: given the pair (c, d), it must be infeasible that any party could generate alternative data x' and a decommitment value d' which match the original commitment value c.

So, for example, we could set $c = hash(N \parallel x)$ for a strong random nonce N and $d = N \parallel x$. It may not be necessary to incorporate x into d if the receiving party is already supposed to know x.

In some cases it is helpful to split x into two parts $M \parallel R$ where M is some message and R is a short random string. In the following protocol due to Vaudenay, this is done, and the commitment is implemented in terms of hashing:

Vaudenay one-way authentication protocol, [23]
1. $A \longrightarrow_N B : INFO_A, hash(INFO_A, R_A, N_A)$
2. $B \longrightarrow_N A : R_B$
3. $A \longrightarrow_N B : N_A, R_A$
4. $A \longrightarrow_E B : R_A \oplus R_B$

Here, R_A and R_B are short (weak) random values of say 16 to 20 bits, whereas N_A and N_B are long random nonces as required in a probabilistic commitment scheme explained above. $\oplus$ is bit-wise exclusive-or.

The intention of this protocol is to assure B that the message $INFO_A$ is actually what was sent to him by A. The role of R_B is the hardest thing to understand about this protocol, so we will simplify it by assuming that there is also an empirical channel from B to A:

Simplified commitment scheme protocol
1. $A \longrightarrow_N B : INFO_A, hash(INFO_A, R_A, N_A)$
2. $B \longrightarrow_E A$: 1-bit committed signal
3. $A \longrightarrow_N B : N_A$
4. $A \longrightarrow_E B : R_A$

It is relatively straightforward to see that this protocol is secure since until A sends N_A, the intruder cannot search for the value of R_A, and so cannot send an alternative Message 1 which has the same R_A in it.

Or, more precisely, if an intruder substitutes Message 1 by $INFO_I, hash(INFO_I, N_I, R_I)$ where R_I is chosen at random, and then substitutes Message 3 by N_I, then there is exactly $\frac{1}{M}$ chance that R_A and R_I will agree, where M is the size of the space from which these values are taken (2^b if they are arbitrary b-bit strings). If this agreement happens, then B will accept $INFO_I$ as having come from A.

In any application one needs to make M large enough to make this 'single lucky guess' attack acceptably unlikely. What we will seek to verify of them is that an intruder capable of searching

through the space of weak values cannot *guarantee* to break the protocol, as could be done if we weakened the above protocol by removing the empirical *committed* message or replacing it by one over a Dolev-Yao channel. For in either case (in the latter via the intruder forging the *committed* from B) A can send both the initial message and N_A before B has received anything. Once the intruder has Message 1 and N_A, it can try each possible value of R_A successively until it finds the right one – namely the one that combines with $INFO_A$ and N_A to give the right hash.

Having done this the intruder can send B (pretending to be A) the alternative Message 1: $INFO_I, hash(INFO_I, N_I, R_A)$, for any values $INFO_I$ and N_I it chooses, and then similarly sends N_I. When A sends R_A correctly to B, this will seem to confirm that she sent $INFO_I$.

It is easy to change the Simplified protocol above into one in which A broadcasts to many B's: all A has to do is wait until she has received the empirical *committed* message from each of them before broadcasting N_A and then sending each the empirical R_A. One can imagine that A might empirically broadcast R_A, for example by reading out the value to a room-full of people holding the other devices involved. Vaudenay's original protocol is not so easy to generalise to a group, but does have the substantial advantage of not requiring the separate empirical commitment message.

Note that both Vaudenay's and the simplified protocol seek to prove to B that he has hashed the same $INFO$ as A. In these protocols he compares the result of hashing his $INFO$ (together with other things) against a value he has received that purports to be what A created when calculating the same hash. These two protocols succeed, where the simplified one without *committed* fails, because they deny the intruder the opportunity to be confident that B will accept anything other than the genuine hash from A to compare against his own. The actual value transmitted on the empirical channel is not itself the hash.

An alternative approach is to make the value compared the hash itself. However it will have to be a much shorter value than a traditional cryptographic hash. The question is whether a shorter hash can be used safely in the presence of combinatorial search.

Our next example is a protocol where this creates a problem:

Naive short hash agreement protocol
1. $A \longrightarrow_N B : INFO_A, N_A$
2. $B \longrightarrow_N A : N_B$
3. $A \longrightarrow_E B : shorthash(INFO_A, N_A, N_B)$

with the protocol succeeding if the value computed by B for the short hash co-incides with the one sent empirically by Alice.

If a full-length hash were sent and checked, this protocol would be secure (even without the nonces) because the fact that A has sent B an empirical message means she has sent one over $\longrightarrow_N$, and the fact that no collisions are assumed possible for long hashes means that the one received first must be the same as the hashed once, given that the hash value is checked. But (we are assuming) this is not an option, and so we need to consider this protocol with a short hash.

Without the nonces, the attacker could prevent the first message getting through, search for any other $INFO'$ that collides with $INFO_A$ under $shorthash$ and send that instead. Adding just N_A would prevent the intruder from doing this searching off line[1] in the event that either the intruder can influence what A sends or A might repeat messages, but in fact gives it an additional opportunity for an on-line attack since it can now choose whatever $INFO'$ it likes and search for N' such that

$$shorthash(INFO_A, N_A) = shorthash(INFO', N')$$

[1]We can characterise an off-line attack as one where much of the work – and in particular the searching – can be done outside the period when the trustworthy participants are active. An on-line attack is where most of the work is done during this period.

Adding Message 2 with N_B prevents this, since the value of the final hash is not determined at the point where B receives Message 1. However, the intruder can still substitute whatever $INFO'$ it likes and now replace N_B in Message 2 by N' such that

$$shorthash(INFO_A, N_A, N_B) = shorthash(INFO', N_A, N')$$

This problem can be fixed by using a similar sort of delayed knowledge as in the first protocol. We give two versions of this which bear an obvious resemblance to the two commitment scheme protocols. In the first, the intruder is denied knowledge of the final hash until B is committed to it, since B it committed to the value of N_A by the hash in Message 1.

One-way HBCK Protocol
1. $A \longrightarrow_N B : INFO_A, hash(N_A)$
2. $B \longrightarrow_E A$: 1-bit committed signal
3. $A \longrightarrow_N B : N_A$
4. $A \longrightarrow_E B : shorthash(INFO_A, N_A)$

In the second, the commitment of B to the final value is indicated by sending N_B. It is not, in this pairwise version of the protocol, necessary to hash N_B, but we include hashing below since it is necessary when this protocol is used to agree values between more than two parties

One-way SHBCK Protocol
1. $A \longrightarrow_N B : INFO_A, hash(N_A)$
2. $B \longrightarrow_N A : hash(N_B)$
3. $A \longrightarrow_N B : N_A$
3. $B \longrightarrow_N A : N_B$
4. $A \longrightarrow_E B : shorthash(INFO_A, N_A, N_B)$

The protocols above are pairwise versions of the Hash Commitment Before Knowledge (HBCK) protocol and Symmetric HCBK protocol, introduced by Nguyen and Roscoe in [13, 14]. As we will discuss in Section 7, it is sensible to strengthen the SHCBK version above by replacing the hashes in Messages 1 and 2 by $hash(A, N_A)$ and $hash(B, N_B)$ respectively. However in this particular version this strengthening does not seem to be strictly necessary.

The properties required of this short hash function are completely different from conventional cryptographic hash functions. In fact it is better to structure it as a *keyed digest* function $digest(hk, INFO)$ where $INFO$ is the substantive information that the protocol seeks to authenticate, and hk is a quantity that the participants seek to randomise. The specification of *digest* is that, for a specified $\epsilon > 0$ and all $I \neq I'$, the probability as hk varies uniformly over its range that $digest(hk, I) = digest(hk, I')$ is no more than ϵ. As discussed in [12, 14, 15], for a b-bit digest function the best-possible ϵ is 2^{-b} and there are various constructions that achieve close to this. So in the usual presentations of the above protocols, the values compared are respectively

$$digest(N_A, INFO_A) \qquad \text{and} \qquad digest(N_A \oplus N_B, INFO_A)$$

where $\oplus$ is bit-wise exclusive or.

3 Approaches to verification

The probabilistic specification of *digest* given above suggests that we might want to check the correctness of this class of protocol on some stochastic tool: verifying that no matter what the intruder does, it cannot increase its chance of success to greater than the ϵ plainly available.

To do this we would need to reformulate the models of protocols and attackers in tools capable of stochastic calculations such as Prism [1] and Apex [7]. This would be desirable not only for the present class of protocols but also for many other security applications. It will be interesting to see if the tools presently available are capable of this sort of analysis, particularly given the large, but not unbounded, number of times we might expect an intruder to be able to try guesses in combinatorial search.

This is not, however, the approach we have taken. Rather, we have tried to embed the positive (for the intruder) consequences of searching into the standard CSP model for cryptoprotocol analysis as set out in [10]. So rather than give the intruder's optimal likelihood of an attack succeeding will give a yes or not answer: if we presume that our intruder has the ability to succeed when performing certain sorts of searching, does it have a deterministic strategy to break security?

In the rest of this section we summarise the standard CSP model that we will later modify.

CSP is a notation for describing and reasoning about interacting systems. It is therefore natural to use it for reasoning about security, where systems are used by two or more parties, one of whom is an imagined intruder, and we have to understand what potential there is for certain types of undesired interaction to occur. In cryptographic protocols, CSP models typically contain representations of each trustworthy participant in a protocol run, any trusted third party that is used, and the intruder. The art of creating such models in the context of strong cryptography is very well developed [8, 18, 21]. Indeed Lowe, with the assistance of others, has created the Casper tool [9] that writes CSP scripts automatically based on a protocol description and a few directives to guide the type of model wanted. These models are then run on the general-purpose CSP refinement/model checker FDR; if Casper is used it interprets any counter-example generated by FDR in the language of protocols.

One of the decisions that has to be taken in these directives is whether to create a model that tests a small instance of a protocol, where only a few identities are present that can run the protocol a very small number of times each, or to use more sophisticated techniques that attempt to prove an arbitrary implementation of the protocol. When checking a new protocol one will probably first do the former, since experience shows that this almost always finds any attack there is, and only if none is found attempt the general result.

Since, in fact, the model check of the general case can only consider a small number of agents and a small number of cryptographic values such as keys and nonces, the general model works by creating a *simulation* of what happens in reality. These simulations are designed so that any attack in the real world is present. This is occasionally at the expense of finding *false attacks*, in other words behaviours introduced by the way the simulation is done rather than being present in the real world. In other words, our finite-state simulation of the infinite real world contains a representation of every real-world trace, and also a few that are not there at all: it over-approximates. For details of the approach used for building proofs of protocols, see [8, 18, 21].

For the time being we will confine ourselves to building the type of CSP model that considers only a small protocol implementation. We will, however, draw a lesson from the above in the sense that we will feel free to over-approximate the behaviour of our intruder since it turns out to be significantly more efficient than attempting to model it precisely. The argument for doing this is the same: it is better to have a model that finds occasional false attacks rather than one that misses real ones.

In CSP models, all the data values in the protocol are drawn from a symbolic recursive data type in which objects such as keys and nonces are primitive constants. For example, for protocols with symmetric and asymmetric encryption, and strong hashing, we might use the type[2]:

[2]Pieces of CSP written in `this font` are quotations from the CSP$_M$ machine-readable CSP language which is a

```
datatype Fact = Sq.Seq(Fact) | Encrypt.Fact.Fact |
                PKE.Fact.Fact | Hash.Fact |
                Alice | Bob | Cameron |   -- agents
                Na | Nb | Nc |  -- nonces
                PKa | PKb | PKc | -- public keys
                SKa | SKb | SKc | -- secret keys
                AtoB | BtoA | Cmessage -- message contents
```

Operations such as hashing, encryption and decryption are thus modelled symbolically, and a finite number of constants are introduced to represent things like node names, keys and nonces. It is natural in such a model to make the strong encryption hypothesis: that there are no hash collisions or other unexpected equalities between terms, and that encryptions are only decipherable by a party in possession of the appropriate key. For of course all members of this type with distinct construction are in fact distinct, and we have no opportunity to analyse the actual encryption methods for vulnerabilities.

Trustworthy agents are modelled as processes that simply engage in the series of messages that they perform in the protocol under consideration, introducing values and performing tests on the coherence of the data concerned. The latter tests are simply coded by saying that only messages which are consistent with previous data will be accepted. Thus any attempt to communicate data that is not thus coherent to such an agent will result in the agent not making progress. The type of attack our model looks for are ones in which trustworthy agents can be led to insecure states. In the case of authentication protocols these are typically states where an agent has completed the protocol in such a way that it leads it to incorrect conclusions about the data it has received or the states of other agents. In our example files this is specified by building trustworthy agents `Alice` and `Bob`, building `Alice` in such a way that she always sends a particular message to `Bob` and trying to establish that he ever thinks she has sent him a different one.

So, for example, we could code the sending and receiving behaviour of the Vaudenay protocol thus:

```
Send(id,ns,rs) =
(rs!=<> and ns!=<>) &
  [] a:diff(agents,{id}) @
     comm.id.a.Sq.<mess(id,a),
        hash(Sq.<mess(id,a),head(rs), head(ns)>)> ->
    ([] r:shortstring @
        comm.a.id.r ->
        comm.id.a.head(ns) ->
        commE.id.a.xor(head(rs),r) ->
     User(id,tail(ns),tail(rs)))


Resp(id,ns,rs) =
(rs!=<> ) &
 [] a:diff(agents,{id}) @
 ([] m:sessmess, r:shortstring,n:nonces @
    comm.a.id.Sq.<m,hash(Sq.<m,r,n>)> ->
    comm.id.a.head(rs) ->
```

mixture of CSP and Haskell-like functional programming as in this defined data type.

```
comm.a.id.n ->
commE.a.id?w:message4 ->
testeq.w.xor(r,head(rs)) ->
if ok(id,a,m) then
User(id,ns,tail(rs)) else ERROR)
```

The process `User(a,ns,rs)` is just the choice between these two.

In the usual CSP model, based on the so-called Dolev-Yao model, the intruder is presumed capable of overhearing, blocking, modifying and faking messages between trustworthy participants limited only by its inability to break cryptographic constructions such as encryption and hashing. It is exactly this communication model that we want for the high bandwidth communication medium $\longrightarrow_N$.

In the standard model the intruder is simply programmed as an agent with an initial knowledge – typically all public information and the private information needed to behave as an agent distinct from the trustworthy ones – which can always learn any `Fact` and send fake messages containing any `Fact` it can construct. The susceptibility of agent's communications to being overheard, blocked and faked is a consequence of the way they are put in parallel with the intruder. The CSP renaming operator is used to map each outgoing communication of a trustworthy agent both to one that gets through and to one that goes only to the intruder. Similarly, the incoming ones are renamed both from ones that come from the honest source and ones that appear to be from there though actually coming from the intruder.

The intruder itself is equivalent to `Spy(Known)`, where

```
Spy(X) = say?x:inter(X,Messages) -> Spy(X)
         [] learn?x -> Spy(close(union(X,{x})))
```

where `close(Y)` applies rules from a set of *deductions* representing the things it can do to data such as encrypt, decrypt, hash and form sequences; and `Known` is the set of things initially known to the intruder. Such deductions are of the type `(X,f)` where `X` is a finite set of `Fact`s and `f` is a single one: for example `({k,Encrypt.k.x},x)` represents symmetric key decryption. As described in [8], in practical CSP models this is factored into a parallel composition of one process per learnable fact to make it work efficiently. We will see later how to do this to the extended intruder model developed in this paper.

It is straightforward to reduce the ability of the intruder so that it can no longer fake messages from Bob to Alice, or block ones from Alice to Bob, say. Equally one can introduce multiple channels between the agents over which the intruder has different powers. In our case there is no problem in building a model with distinct channels for $\longrightarrow_N$ and $\longrightarrow_E$, the latter of which cannot be faked. In the example above these channels are written `comm` and `commE` respectively.

If this is combined with a model of the intruder in which *shorthash* is treated in the same way as *hash*, then no attack will be found on any of the example protocols[3], even the variants on The Naive protocol. In the standard model, the only thing that the intruder can do with hash is apply it: if it knows x then it knows $hash(x)$. In other words it has no ability to deduce x from $hash(x)$ or to create $x \neq y$ such that $hash(x) = hash(y)$.

In the next two sections we will see how to give our intruder appropriate versions of these two abilities. In trying to understand what is appropriate it is useful to identify the subset of the symbolic values in `Fact` that are *weak*: ones ranging over domains small enough to make searching attacks meaningful.

[3]If the communications on $\longrightarrow_E$ were replaced by the same ones over $\longrightarrow_N$, then attacks would be found on all the protocols since all of them depend on the fact that communications on $\longrightarrow_E$ cannot be faked.

4 Deducing preimages

Suppose our attacker knows a value $C[w]$ whose construction depends on the weak value w but does not know w itself. Under what circumstances can it test guesses of w? In [10], Lowe addresses a generalisation of this question in which $C[w]$ is replaced by a general value v that is capable of *verifying* w. So for example, in Lowe's work, v might be a piece of data that the attacker requires w to decrypt (which is typical in protocols that bootstrap based on weak, shared, secret passwords). In some very detailed work, Lowe analyses under what circumstances we can say that the attacker has an *alternative* route to deducing v given w that *really* requires w. To achieve this, monitoring the sets of deductions used, Lowe uses a significantly altered intruder model.

We used a simpler and less general approach that is well suited to the case where the verifier v takes the form $C[w]$. Rather than change the nature of the intruder model, we stick to the one in which it is based on deductions $X \vdash f$ for $X \cup \{f\} \subseteq fact$. We add a new deduction, which to all intents and purposes is $\{C[w], C[\cdot]\} \vdash w$: if the intruder knows the result of applying a context to a weak value w, and knows the context itself, then it can work out w.

To implement this we need to understand what it means to "know" a context. Practically, it means that the intruder has the ability, given an arbitrary value s, to calculate $C[s]$. In general an intruder can use other, trustworthy, agents as "oracles" to help it calculate, but this would have two big disadvantages when it comes to performing a combinatorial search. The first is that such interaction would almost certainly be a lot slower than the intruder calculating for itself. The second is that if a trustworthy agent were performing a calculation for every value that the intruder tried, it would raise suspicions that something evil was afoot.

Therefore our default position on knowing a context is that, given s, the intruder must be capable of working out $C[s]$ without any external assistance. We have implemented this by introducing a constant s representing an arbitrary weak member of $Fact$ that is known to the intruder, but such that no $Fact$ involving s can be transmitted across any channel.

Knowing $C[\cdot]$ therefore is equivalent to knowing $C[s]$, so the above deduction becomes

$$\{C[w], C[s]\} \vdash w$$

Thus, given the particular type of combinatorial searching we are discussing in this section, it is possible to augment the standard deduction model to analyse for it in such a way that scarcely changes the overall protocol model.

Of the example protocols, the only ones where there is a secret weak value to guess are Vaudenay's protocol and the variants we discussed. Our modified model can be run on these protocols, and attacks are duly found on the version of the second without the *committed* message and the first protocol if B does not generate the short nonce N_B which is instrumental in the computation of the short authentication string sent in Message 4.

This is therefore, at least within the scope of our examples, a successful way of building searching for a weak secret value into the standard CSP models of cryptoprotocols. It would be interesting to experiment with further examples such as those used by Lowe to see how this method compares with his in more general circumstances.

5 Searching for collisions

In the rest of the example protocols, the secret values used by the agents are not weak. The only weak value used is the short hash or digest, which is already public at the point it is calculated. The nature of the attacks described on the "Naive short hash agreement protocol" and variants of the

other protocols take a different form: rather than searching for a weak secret, the intruder rather looks for different strong values that some weak context (one that generates a weak value). This is a rather different sort of activity to capture in our symbolic protocol model, because rather than discovering an existing value we are expecting our intruder to discover a new one with a special property.

Introducing a new constant into $Fact$ for every special property that might be accessible to the intruder would be completely impractical, even if these were finite in number. For example, for every weak value w and every context $C[\cdot]$ that the intruder knows, whose value depends properly on its argument, it can search for v such that $C[v] = w$. Since it is quite likely that these searched-for constants can be used to create other contexts, it is likely that in most examples the number of symbolic contexts actually available to the intruder will be infinite.

With the exception of the first group of example protocols, which we can already examine thanks to the pre-image guessing techniques, the examples all have the following features:

- The only roles that weak values have is to be compared over the empirical channel: no cryptographic functions are ever applied to them.
- In the pairwise versions of the protocols presented above, only a single pair of weak values are compared. [This is not true of group versions, since then we will typically be comparing values computed at N different locations.]

We conclude that any flaw relating to searching in these protocols will be due to a single pair of computed weak values, computed entirely from strong ones, being equal. Thus, for these protocols, this provides semi-formal justification for the assertion that if the intruder can break the security of the protocol, it can do so with a single search.

To give it this ability we can add a single additional constant where it can choose the equation that the constant satisfies. The target of this search would be a weak value w, and the object would be to find some fresh value x for a weak-valued context $C[\cdot]$ such that $C[x] = w$, where $C[\cdot]$ is a context that the intruder knows.[4] Equality would not actually hold in the symbolic type $Fact$: rather we would need to modify equality tests etc in the CSP model so that it behaved as though the values were equal.

To achieve this, the triple $(x, C[x], w)$ becomes part of the state of the protocol model after the search has been performed. The multiplicity of weak values w and contexts $C[\cdot]$ that the intruder knows means that this typically increases the state space size considerably, making the checks of relatively simple protocols like the (pairwise) examples in Section 2 into lengthy runs. It is this effect that makes it unrealistic to contemplate extending this model of the intruder to perform more than one search.

This recalls the early days of protocol verification using CSP models [8, 18, 21], where the intruder models used were, for similar reasons, restricted to "remember" only one or perhaps two facts over any above their basic initial knowledge. This proved effective in finding interesting attacks, but was obviously inadequate even to prove the security even of small protocol models: we clearly needed to extend the intruder model to have an arbitrary memory. This proved possible with the "perfect spy" [21], now a standard part of all CSP cryptographic protocol analysis. One of the interesting side-effects of this was that the state spaces of checks run with this unbounded

[4] We might note that this does not exclude our intruder model from the effect of performing the "birthday attack" where it solves the equation $C_1[x] = C_2[y]$ for a pair of fresh values x and y, since having introduced one value y and constructed $C_2[y]$ it can search for x. Of course this does not reflect the relative efficiency of this style of attack, but all we are doing is to look for attacks on the assumption that searching does yield a result without reflecting efficiency.

memory model were usually smaller than with a highly restricted memory, because the choice of which thing to remember no longer contributed.

What we would like to be able to do similarly is to find an intruder model that reflected its ability to perform an arbitrary number of combinatorial searches in a single run. We have not solved this problem completely, but have found an approach that reduces the complexity of model checking and provides a less limited intruder.

Our new model has no explicit representation of search, but rather asks each time two weak values are compared whether the intruder *could* have performed a search to force them to be equal.

To achieve this, we allow the protocol and intruder model to proceed normally, except that whenever the intruder invents a fresh value it takes a snapshot of its memory. Then, whenever a comparison of weak values occurs, we ask whether the search could have happened in the state that generated the most recent fresh value created by the intruder.

This of course will also have an effect on the state space: simply on the grounds of the size of $Fact$ one cannot introduce more than a very few fresh constants, and there will clearly be an overhead from the extra memory information. As we will see, however, this overhead is much lower than from the model in which we record a triple recording the search that was performed.

Initially we will consider the case in which the intruder is allowed to search for a single fresh value with a special property. We will need a revised intruder process that keeps a record of its own memory at the point when the search was performed, and which can judge whether the equality of two weak values could have been the result of that search. The sequential process we want is:

```
Spy0(X,F) = say?x:X -> Spy0(X,diff(F,explode(x)))
            [] learn?y:No(F) -> Spy0(close(union(X,{y})),F)
            [] search?f:F -> Spy1(X,X,f)

Spy1(X,Y,f) = say?x:X -> Spy1(X,Y,f)
              [] learn?y -> Spy1(close(union(X,{y})),Y,f)
              [] testeq?{x,y}:equals(Y,f) -> Spy1(X,Y,f)

equals(Y,f) = union({{x,y} | x,y <- inter(WeakComp, Y), member(f,explode(y))},
                    {(x,x) | x <- WeakComp})
```

In other words, unless the two values are actually the same, one of them –here `y`– must involve `f`, and at the point `f` was introduced these two facts were known to the intruder. Note that at that point, knowing `x` and `y`, if they involve `f`, is equivalent to knowing the contexts used to create them so we do not need the special placeholder `s` to model this type of searching.

This definition can be simplified by only having the intruder record the point-of-search knowledge of *weak* facts, since its knowledge of other data then is irrelevant to its subsequent behaviour.

Just as with the classic intruder model, this will not run on FDR thanks to the state space of this sequential process. We therefore need to factor this adapted model in a similar way: with one parallel process for each learnable fact. In the classic model this component process is one of two states `Ignorantof(x)` or `Knows(x)`. These will still suffice for the components that do not represent weak values, since these are not required to compute the `equals` relation. Such processes do not need to synchronise on the `search.f` action, bearing in mind that there will be no intruder component for the fresh fact `f` on which the search is performed since for the purposes of coding this is in the knowledge of the intruder and is not in this sense *learnable*. So we will need modified components only for weak `x`.

For these, processes similar to these two are required prior to the `search`, while after it there will be three states:

- One in which the fact `x` is still not known: `IgnorantAfter(x,f)`
- One in which `x` was known at the point of the search (and is therefore still known): `WasKnown(x,f)`.
- One in which `x` is known now but was not known at the point of the search: `RecentlyKnown(x,f)`.

Note that these three states also record the fact that was searched on, since this will be needed to determine the modified equality relation. The need for this parameter can be dropped if we split `WasKnown(x,f)` into two states: one for the case where `f` contains the searched-for value, and one for the case where it does not:

In the following definitions the type of `testeq` has changed to be two separate comparable components rather than a set. By fixing the second of these components to contain the search value, we can ensure that one of the two "equal" values always does contain this.

- The first state does not know its fact, before the search.

```
ignorantof(f) = member(f,SpyMightHear) & learn.f -> knows(f)
                [] infer?t:infs(f) -> knows(f)
                [] search?x -> ignorantafter(f)
                [] member(f,WComparisons) & testeq.f.f -> ignorantof(f)
```

- The second state does know its fact, before the search. Note that the result state after `search.x` depends on whether it contains `x` or not.

```
knows(f) = member(f,SpyMightHear) &  say.f -> knows(f)
           [] member(f,SpyMightHear) & learn.f -> knows(f)
           [] infer?t:deds(f) -> knows(f)
   [] member(f,Banned) & spyknows.f -> knows(f)
           [] member(f,WComparisons) & testeq.f.f -> knows(f)
           [] search?x -> (if member(x,atoms(f)) then wasknownwith(f)
                                                 else wasknownwithout(f))
```

- The state where `f` is not known after the search.

```
ignorantafter(f) = member(f,SpyMightHear) & learn.f -> recentlyknown(f)
          [] infer?t:infs(f) -> recentlyknown(f)
          [] member(f,WComparisons) & testeq.f.f -> ignorantof(f)
```

- The state where `f` is known after the search and involves the search constant.

```
wasknownwith(f) = member(f,SpyMightHear) &  say.f -> wasknownwith(f)
           [] member(f,SpyMightHear) & learn.f -> wasknownwith(f)
           [] infer?t:deds(f) -> wasknownwith(f)
   [] member(f,Banned) & spyknows.f -> wasknownwith(f)
           [] member(f,WComparisons) & testeq?x!f -> wasknownwith(f)
           [] member(f,WComparisons) & testeq!f?x -> wasknownwith(f)
```

- The state where `f` is known after the search and does not involve the search constant.

```
wasknownwithout(f) = member(f,SpyMightHear) &  say.f -> wasknownwithout(f)
           [] member(f,SpyMightHear) & learn.f -> wasknownwithout(f)
           [] infer?t:deds(f) -> wasknownwithout(f)
   [] member(f,Banned) & spyknows.f -> wasknownwithout(f)
           [] member(f,WComparisons) & testeq!f?x -> wasknownwithout(f)
```

- The state where `f` was not known at the point of search but has been learned since.

```
recentlyknown(f) = member(f,SpyMightHear) &  say.f -> recentlyknown(f)
           [] member(f,SpyMightHear) & learn.f -> recentlyknown(f)
           [] infer?t:deds(f) -> recentlyknown(f)
   [] member(f,Banned) & spyknows.f -> recentlyknown(f)
           [] member(f,WComparisons) & testeq.f.f -> recentlyknown(f)
```

Above, `infs(f)` are the inferences that allow `f` to be deduced, namely ones of the form `(X,f)`, and `deds(f)` are ones that `f` contributes to, namely ones `(x,f')` where `f` is in `X`.

Note that the channel `equal` is treated asymmetrically here: the first component can be communicated whenever the fact had been known at the point of the search. The second can be communicated whenever both this happens and the fact actually involves the value on which the search occurred, or the two components are equal without the need to search. This ensures that the conditions involved in the definition of `equals(z,Y)` above are true when the event `equal.x.y` occurs after a search: both must have been known at the point of the search and at least one of them (i.e. `y`) involves the fresh value introduced at that point.

Note that the *only* point at which the identity of `f` is required in the above definitions is in `WasKnown(x,f)` in determining the second component of `equals.x.y`. This suggests that a more concise definition in terms of FDR compilation will be produced by modifying the above definitions, deleting the parameter `f` from the last three states and dividing up the final state into two: `WasKnownWithf(x)` and `WasKnownNof(x)` with the decision about which of these two to enter being made at the point of the search. So the definition of `Knows(x)` becomes

```
Knows(x) = member(x,Messages)&learn.x -> Knows(x)
           [] infer?d:deds(x) -> Knows(x)
           [] member(x,Messages)&say.x -> Knows(x)
           [] search?f:Searchable ->
               if member(f,explode(f)) then WasKnownWithf(x)
               else WasKnownNof(x)
```

The resulting components are synchronised on relevant `infer` events as in the classic intruder model. The weak components are synchronised on all `search` events, and `equals.x.y` is in the alphabets of intruder components `x` and `y`.

The classic intruder only has a component for each `LearnableFact`, namely a relevant member of the type `Fact` that is not known initially to the intruder but which can be learned. There is an additional one-state process `SpyKnows` that performs its functions for the facts that are in the initial knowledge of the intruder. More-or-less the same will do in our revised case except that the corresponding process now has to synchronise on `search` events, allowing one of them, and after that synchronises with appropriate `equals` events.

```
SpyKnows = learn?x:IntruderInitialKnowledge -> SpyKnows
           [] say?x:IntruderInitialKnowledge -> SpyKnows
```

```
            [] equal?x:IntruderInitialKnowledge!x -> SpyKnows
            [] search?f -> SpyKnows'({w | w <- WeakComp, member(f,explode(w))})

SpyKnows'(Y) = learn?x:IntruderInitialKnowledge -> SpyKnows'(Y)
            [] say?x:IntruderInitialKnowledge -> SpyKnows'(Y)
            [] equal?x:IntruderInitialKnowledge!x -> SpyKnows'(Y)
            [] equal?x?y:Y -> SpyKnows'(Y)
```

One thing that is not captured by the above definitions (either the parallel one or the sequential `Spy0`) are the practical constraints of searching for a value `f` with a specific property. In reality the value comes into being only when it is searched for, and so the value certainly cannot have been seen outside the intruder at the point the `search.f` event occurs. So we need an additional process running in parallel with the intruder that only allows `search.f` at points where the intruder has never communicated `say.x` for any `x` such that `member(f,explode(x))`.

This can be incorporated into the above definitions at the expense of some more complexity, or we can run another process in parallel to enforce it, which is probably more straightforward to explain. The intruder will have one or more constants representing fresh values that it can search for, and in the model we are now considering it can perform a single search for a value of one of these. These constants are placed in the initial knowledge of the intruder process. We can regard the values of such constants as formal and undetermined until the moment they or something based on them is seen by another agent, after which they become fixed: there is at least a superficial analogy with quantum information theory here. To enforce this we could run the following process in parallel with the intruder for each searchable constant `f`, synchronising on the set of events it uses:

```
SearchReg(f) = search.f -> NoSearch(f)
               [] say?x:{x | x <- Messages, member(f,explode(x))} -> NoSearch(f)

NoSearch(f) =  say?x:{x | x <- Messages, member(f,explode(x))} -> NoSearch(f)
```

This is not, however, the most efficient model. To understand this, contemplate the possible timings of a particular search relative to when the searched-for value is used by being sent (via the `say` event in the definitions above) to a trustworthy agent.

The longer it waits to make this search before this send, the better for the intruder, because the more values and contexts it knows for the search to be based on, and the larger the set of values that `Set1(X,Y)` records as `Y` making more pairs "equal". In fact if any attack is possible for a search at any time, it is also possible if the search happened a notional instant before the send. (Note that the actual time required to do the search is irrelevant to our symbolic model.)

It therefore makes sense in principle to identify the point of the search with the first time a value involving the searchable constant is sent. By doing this we potentially eliminate many states where the search has been performed but the value not yet used. CSP enables us to do this by introducing a combined channel `searchandsay.(f,x)` where `f` is a searchable constant and `x` is a `Message` that involves it. Getting the right events to synchronise using this approach is, however, complex. We therefore adopt the much simpler solution of stipulating that, in the system as a whole, every time a `search.f` event occurs, the next event has the intruder perform `say.x` for an `x` involving `f`. This can be done by parallel composition (synchronising on all visible events) with a constraining process `SearchAndDeliver`.

Exactly as with the classic intruder model, the parallel composition of all the `ignorantof(f)` processes has the channel `infer` hidden and the FDR operator `chase` is applied. This forces all inferences to occur as early as possible, which has a huge effect in reducing the state space.

The intruder model above has been used to establish the expected results – namely the absence of any attack or the presence of one based on searching – for each of the protocols in Section 2 that are based on the agreement of a final short hash or digest. For example, no attack is found on the version given of the HCBK protocol given there, but if we interchange Messages 2 and 3 – so that A sends na before it receives the *committed* messages – FDR produces traces such as the following one.

```
<take.Bob.Alice.Sq.<BtoA,Hash.Nb>,
take.Bob.Alice.Nb,
search.fN,
fake.Bob.Alice.Sq.<Cmessage,Hash.fN>,
fake.Bob.Alice.fN,
commE.Alice.Bob.Yes,
commE.Bob.Alice.Digest.Nb.BtoA,
testeq'.{Digest.Nb.BtoA,Digest.fN.Cmessage},
error>
```

This represents the intruder blocking and learning `Bob`'s first two messages to `Alice`. Since it then knows `Nb` it can search for a value `fN` such that the equality implied by the `testeq'` event holds, and then fake alternative messages to `Alice` "from" `Bob`.

As described earlier, CSP files can be downloaded which will allow readers to test these protocols for themselves.

5.1 Other approaches to modelling collisions

The model we have developed for collision searching above has two curious, and related properties.

(a) While the constants used for searching are "known" to the intruder from the beginning, the search for the actual value does not occur until part-way through the run (when `search.f` occurs).

(b) Once this search is performed, our model assumes it actually satisfies *all* equations that could have been searched for, not just one.

The latter is certainly, in general, a phenomenon that could lead to the discovery of false attacks that rely on a single constant satisfying multiple equations. This is not possible in the examples we have considered to date since none of these protocols relies on any more than a single equality test, and that test being true does not lead to nodes releasing information that would be useful to the intruder in subsequent runs.[5]

This would be an issue if we used a similar model for the "FlexiMac" protocol of [16] in which a digital certificate for message M consists of a signed set of pairs $(k, digest(k, M))$ where there are N keys chosen at random. The certificate is verified by choosing, at random, $V << N$ of the keys and verifying that the digests of the received M under these keys agree with those in the certificate.

In a simple case $V = 2$ and $N = 1000$, perhaps chosen because it is assessed that there is a good chance that an attacker can find M' such that $digest(k, M') = digest(k, M)$ for k any one of the N keys, a 1% chance that it can find M' and $\{k_1, k_2\}$ such that this equation holds for both k_i simultaneously, and a negligible chance for more k_i simultaneously.

[5] In any case, the example files usually only consider models where no more than one run between trustworthy parties occurs, and therefore usually only one `testeq` can occur on any run.

Since there are $500,500$ different pairs of keys that the verifier might pick at random, there is actually only a one in $50M$ chance that an attack will work. However, letting the attacker search for M' at the time it knows the certificate using the model described above will mean that, symbolically, the fresh fM chosen will "solve" all the equations $digest(k, M) = digest(k, fM)$ simultaneously as k varies.

We could address this particular issue by the addition of an additional restriction to our model: a process that only allows a single set of two distinct values `{d1,d2}` to be communicated on `testeq'` in a single run: once a single such pair has occurred the regulator process will prevent all other ones. This could easily be increased to any fixed number `N` depending on how many different equations we think the intruder can solve simultaneously. In the following definition, `ps` are the "equal" pairs of distinct values observed to date.

```
EqLim(N,ps) = card(ps) < N & testeq'?p:S2 -> EqLim(N,union(ps,{p}))
              [] testeq'?p:union(S1,ps) -> EqLim(N,ps)
```

where `S2` are the two-element sets of comparable weak values, and `S1` are the one-element sets (i.e. representing pairs that are equal without searching).

This approach does not allow satisfactory analysis of the FlexiMac protocol as a whole. Fortunately that is fairly easy to address via probabilistic calculations.

If we set `N=1` in `EqLim(N,{})`, we can refine further the way our understanding of the searched-for value grows as a run progresses. At the start of a run we know nothing about its value; at the point of the search we have a potentially large set of equations it might satisfy; and we know exactly which it satisfies at the point it turns up in a successful equality test of differently-constructed values.

One of the keys to the efficient use of FDR is reducing the number of states that are explored, and it is interesting to contrast the above approach with an early attempt by Roscoe to model the HCBK protocol in 2004, shortly after he invented it. In that the single equation to be solved was chosen (as is natural intuitively) at the point of the search itself. Thus the search event had three parameters: the search constant and the two sides of the equation. This was subsequently stored as a parameter to allow the determination of subsequent equality tests.

In all protocol states after the search this early model therefore kept a separate system state for all the equations that could have been searched over. This is hugely less efficient than our new approach of leaving this equation undetermined until an event that fixes it is executed: now there is just one system state per protocol state.

5.2 Modelling $\oplus$

Both the Vaudenay and SHCBK protocols make crucial use of bit-wise exclusive-or. It is important that we have an efficient representation of it within our CSP models that reflects its algebraic properties if we are going to analyse for attacks that might use its properties. (For examples of such attacks, see Section 7.)

We have modelled the fields that are liable to be xor-ed as objects of the form `Xor.S`, where `S` is a set of constants of the given sort. It is reasonable to assume that the xors of all such collections of a few constants are indeed different, and we can define an operator `xor` that precisely reflects the algebraic properties of $\oplus$:

```
xor(Xor.S,Xor.T) = Xor.union(diff(S,T),diff(T,S))
```

The useful feature of this representation is that there really is just one value in `Fact` for each real value. The `Xor` constructor has the following intruder deductions:

- `(S,Xor.S)`: if the intruder knows every member of the set `S` it can xor them together.
- `({Xor.{f},f)`: if the intruder knows the lifted `Xor` of a fact then it knows the fact.
- `({Xor.S,Xor.T},xor(Xor.S,Xor.T))`: if the intruder knows two facts it can xor them together.

This representation has proved very successful in dealing with relevant protocols mentioned in this paper: the reader will find it in the example files that accompany this paper. It is somewhat different from the representation of xor used in Casper, and homeomorphic to one previously proposed by [13].

The only potential problem is that the size of the space of `Xor.S` grows exponentially with the number of constants available for `S`, and this in turn could well have an explosive effect on parameters such as the overall alphabet of the model and the number of deductions.

6 Combined approach

In the previous two sections we have described techniques by which FDR models can be extended to allow the intruder to search for weak values that are concealed within known contexts, and also search for collisions between constructed contexts that are weak at the top level.

There is nothing that prevents the two being used together in a single model, but we are not aware of any published protocol for which they are both relevant. Consider, however, the following protocol, in which it is assumed that A and B share a weak, supposedly secret, password P_{AB}.

$A \longrightarrow_N B : M$
$B \longrightarrow_N A : Nb$
$A \longrightarrow_E B : digest(P_{AB} \parallel Nb, M)$

Here, M is the intended messages and Nb is a nonce.

Suppose the intruder has observed one run of this protocol. Because it knows M and Nb it can search for the value of P_{AB} as in Section 4. When a second message M_2 is sent from A to B, the intruder can substitute it by the message M' of its choice, trap the nonce Nb of this second session, and search for Nb' such that

$$digest(P_{AB} \parallel Nb, M') = digest(P_{AB} \parallel Nb', M_2)$$

and send this to A in Nb's place.

7 Group protocols

A number of HISPs (including the original versions of HCBK and SHCBK) have been developed which allow an arbitrary-sized group to authenticate data from one or all of them based on agreement about a single weak value (a digest in these two cases).

There are no new powers that the intruder needs in order to cope with group protocols, so the only real issue is how to model the execution and correctness of a group protocol sufficiently efficiently to make analysis feasible.

We will use two group versions of the SHCBK protocol as our examples. The S in its name stands for "Symmetric", and all its messages are broadcasts from each node to every other. When two of us initially discovered this protocol presented it at a workshop [13] the protocol took the form:

Symmetrised HCBK protocol (SHCBK), [13]		
1.	$\forall A \longrightarrow_N \forall A'$:	$A, INFO_A, hash(k_A)$
2.	$\forall A \longrightarrow_N \forall A'$:	k_A
3.	$\forall A \longrightarrow_E \forall A'$:	$digest(k^*, INFOS)$
		where k^* is the XOR of all the k_A's for $A \in \mathbf{G}$

However when we subsequently published this work in a journal [14] it had become

Symmetrised HCBK protocol (SHCBK), [14]		
1.	$\forall A \longrightarrow_N \forall A'$:	$A, INFO_A, hash(A, k_A)$
2.	$\forall A \longrightarrow_N \forall A'$:	k_A
3.	$\forall A \longrightarrow_E \forall A'$:	$digest(k^*, INFOS)$
		where k^* is the XOR of all the k_A's for $A \in \mathbf{G}$

In each case, any pair of nodes who successfully compare their digests on the final step are supposed to know that their views of the various $INFO_A$s (including their own) coincide.

Note that the only difference is that the shares hk_A of the final digest key are hashed alone in the first version and with the names of their originators in the second.

This modification was made because we were aware that the use of $\oplus$ in combining these shares made a reflection attack available to the intruder whereby it could adopt another node's hk_A as the one belonging to another node without actually knowing it. Once it knows $hash(hk_A)$ it can send this two whoever it likes, pretending it is from any one of the nodes, even though it does not yet know the value of hk_A. Once A reveals the value of hk_A in Message 2, the intruder can copy this value to follow up all the sends it has made of its hash.

Since the shares of hk are combined by $\oplus$, this has the effect of selectively cancelling some of the shares hk_A in the calculations of some or all B.

The modification of replacing $hash(hk_A)$ by $hash(A, hk_A)$ immediately removes this possibility of reflection (on the assumption that nodes expect all their partners to have distinct identifiers A, which is in practice necessary in group protocols where messages are broadcast). For at the point where the share hk_A is sent openly in Message 2, apparently from $B \neq A$, the check of the hash $hash(B, hk_A)$ will not agree with the reflected hash $hash(A, hk_A)$ from Message 1.

All this was clear to us when we made the modification, but we did not investigate the extent to which failure to make it could compromise security. Certainly, security is not always compromised: note that one of the examples earlier in the present paper is a pairwise adaptation of Version 1, and we have successfully verified it.[6]

One of the most interesting properties of SHCBK is that, if run by a group G, it authenticates two trustworthy parties and their data to each other even if all of the other members of the group are corrupt and under the control of the intruder.

As a natural step up from two-user sessions, we have therefore built models of this protocol with the usual two trustworthy participants Alice and Bob, who can either run the protocol with each other in a group of two or with a third corrupt identity in a group of three. The specification is that, provided Alice and Bob agree on the final digest, their information is correctly authenticated to the other.

[6]The version quoted in Section 2 is not susceptible to the reflection attack because it does not xor the two shares. However using $digest(Na \oplus Na, M)$ instead of $shorthash(Na, Nb, M)$, as is done in the CSP example file for this protocol, does admit the attack but does not change the verifiability. In fact we have only verified a small implementation of it, but there is every reason to believe that this pairwise version remains secure in arbitrary-sized implementations.

Since the protocol model we build is symmetric, with no pre-set order of communication other than that each node completes its activities in Message 1 before Message 2, and similarly completes Message 2 before engaging in digest comparisons in Message 3, we can restrict our attention to showing that Alice's information is correctly authenticated to Bob. This permits a significantly reduced state space for the component processes.

We have to be very careful with state space with more participants in the protocol, because the number of constants in a model grows, the size of the messages (in this protocol Message 3 which is the digest of one value by another, each of which grows with the number of participants) grows very quickly, as do the parameters and therefore the state-space sizes of the individual agent processes.

The definition of a trustworthy agent for the first version is as follows. Instead of sending and receiving a fixed sequence of messages, it goes through a phase for sending and receiving each of the three messages of the protocol that it has to exchange with all the other parties.

```
User(id,ns) = ns!=<> &
  session?S:sessions(id) ->
 (Phase1(id,xn(head(ns)),{},S);User(id,tail(ns)))

Phase1(id,n,bs,S) =
  comm.id?a:diff(S,{id})!Sq.<mess(id),hash(n)> ->
                                Phase1(id,n,bs,S)
[]([]a:rest(S,id,bs),m:sessmess, n':xnonces@
      comm.a.id.Sq.<m,hash(n')> ->
      Phase1(id,n,union(bs,{(a,m,n')}),S))
[] rest(S,id,bs)=={} & Phase2(id,n,bs,{},n,S)

Phase2(id,n,bs,cs,hk,S) =
  comm.id?a:diff(S,{id})!n -> Phase2(id,n,bs,cs,hk,S)
[]([] a:diff(S,union({id},cs)) @
       comm.a.id.checknonce(a,bs) ->
       Phase2(id,n,bs,union(cs,{a}),
              xor(hk,checknonce(a,bs)),S))
[] diff(S,union({id},cs))=={} &
   let d=digest(hk,summary(S,id,n,bs))
   within Phase3(id,d,bs,S)

Phase3(id,d,bs,S) =
SKIP
[] ([] a:diff(S,{id}) @ commE.id.a.d ->
                          Phase3(id,d,bs,S))
[] commE?a:diff(S,{id,Cameron})!id?w:WComparisons ->
   testeq.w.d ->
   (if ok(id,a,msg(a,bs)) then Phase3(id,d,bs,S)
                                  else ERROR)
```

Note that in each of the first two phases it broadcasts its own message arbitrarily often without recording who it has been sent to, but carefully gathers the information it has been sent.

This protocol gave the following trace representing an attack: `Cameron` has been able use the reflection attack outlined above, adopting `Bob`'s hk as his own. This means that he can already

know what final digest key Bob will use even before Bob has revealed its own Nb: the properties of $\oplus$ means that that share of the key is nullified in the operation of both Alice and Bob. A search is therefore performed before an intruder-modified copy of Alice's Message 1 is delivered to Bob when it has access to the values that both Alice and Bob will generate for their final comparison.

```
<session.{Cameron,Alice,Bob}
take.Alice.Bob.Sq.<InfoA,Hash.Xor.{Na}>
comm.Bob.Alice.Sq.<Cmess,Hash.Xor.{Nb}>
fake.Cameron.Alice.Sq.<Cmess,Hash.Xor.{Nb}>
take.Alice.Cameron.Xor.{Na}
fake.Cameron.Bob.Sq.<Cmess,Hash.Xor.{Nb}>
search.fN
fake.Alice.Bob.Sq.<Cmess,Hash.Xor.{fN}>
comm.Bob.Alice.Xor.{Nb}
fake.Cameron.Bob.Xor.{Nb}
fake.Alice.Bob.Xor.{fN},
fake.Cameron.Alice.Xor.{Nb}
commE.Alice.Bob.Digest.Xor.{Na}.
  Summary.{(Alice,InfoA),(Cameron,Cmess),(Bob,Cmess)}
testeq'.{Digest.Xor.{fN}.
           Summary.{(Alice,Cmess),(Cameron,Cmess),(Bob,Cmess)},
         Digest.Xor.{Na}.
           Summary.{(Alice,InfoA),(Cameron,Cmess),(Bob,Cmess)}}
error>
```

Because of the way our system is plumbed together, each comm action is a communication between the two trustworthy agents that is overheard by Cameron, fake.X.Y is a communication from the intruder to Y that is apparently from X, and take.X.Y is a communication from X to Y which does not get through if Y is trustworthy, and is in any cawse heard by the intruder Cameron.

The net result of the above trace is that Bob thinks that Alice has sent the information Cmess, whereas in fact she has send InfoA. Note that the representations of the digest keys and shares of them follows the model of xor we discussed in Section 5.2. The fact that all the Xor-ed sets above are singletons above is a consequence of the way this particular attack works. Normally, of course, we would expect the final key used in a session involving k agents to be the Xor of a set of k values.

At the point of the search.fN the intruder has the knowledge to find a value which makes the equation implicit in the testeq' message true.

Note that this trace does not, thanks to our broadcast model, include all the sends of of Messages 1 and 2 by Alice and Bob. They could be inserted without destroying the attack, but our Cameron (the intruder) does not need all these sends either to learn the contents of the messages (at most one send by one of them suffices) or to proceed (because he is not constrained to follow any protocol).

Fortunately, attacks disappear for Version 2: the corresponding file for it yields a successful check for the authentication specification, showing that the protocol does indeed authenticate A's information to B even when C is corrupt.[7]

This particular attack clearly exploits the intruder's ability to participate in the protocol as a participant. Note that even if there is a real Cameron who is trustworthy, there is nothing to prevent the intruder taking his role in the first two messages of this protocol, so unless Alice and/or

[7]What that means is that if A and B agree on the digest, then A and B agree on what A's information is. The intruder controlling C can, if it wishes, prevent them from agreeing in various ways.

Bob check agreement of the digest with the real Cameron there is nothing to prevent the intruder taking his place from the perspective of the other two users.

Even though the pairwise version of the protocol in Section 2 is secure, FDR succeeds in finding attacks on the group version of the protocol for sessions involving just the group `{Alice,Bob}`. The explanation of this apparent contradiction is the more liberal communication order in the group version of the protocol. In it, all participants are prepared to send Message 1 before receiving a Message 1 from any other party. However in the one-way pairwise version the responder/receiver B will only send a hash of his own key share after receiving Message 1 from A. That difference is crucial.[8]

Running our model on different variants of this protocol is an excellent demonstration of the power of the CSP model we are proposing. It also emphasises the advisability of following normal commonsense protocol design practices such as explicitness of encrypted/hashed components in designing HISP-style protocols, just as in more traditional ones.

8 Prospects for generalisation

For some protocols it is to draw a distinction between weak values that are susceptible to straight combinatorial search and semi-weak values that are too long for this but are vulnerable to the birthday attack. In other words it might be impractical to search for v such that $C[v] = sw$ where sw is a semi-weak value (perhaps 80 bits), but feasible to search for v such that $C[v] = C'[v]$ where C and C' are both semi-weak contexts.

Consider, for example, the simple protocols in which A sends $Info_A$ to B backed up by an empirical communication of $hash(Info_A)$, and where A sends $Info_A, N_A$ to B backed up by $hash(Info_A, N_A)$. In some circumstances it might be possible for the intruder to influence what A sends as $Info_A$, but we can assume that N_A is always chosen randomly. (Note that these protocols were both discussed as weak versions of the Naive protocol in Section 2.)

For the first protocol, the intruder can search for distinct $Info_A$ and $Info'_A$ that hash to the same value before influencing A accordingly. This is an instance of the birthday attack and is much easier for the intruder than attacking the second protocol where, since it cannot influence the choice of N_A, it simply has to search for N'_A such that $hash(Info_A, N_A) = hash(Info'_A, N'_A)$. It is natural that there will be lengths of hash that can be considered effectively secure for this second protocol but not the first.

We could easily allow for types that we know to be semi-weak in a protocol description (either instead of, or as well as, weak ones) by *not* allowing the deduction of such a value v from $C[v]$ where C is a known context, and *only* allowing the search for values v solving equations between semi-weak contexts when *both* sides depend on v. The latter can be achieved by altering the way the *testeq* channel is used.

We have to recognise, however, that our model as introduced so far has limitations. In particular we have only considered small implementations of our protocols, with small groups only able to run protocols a very limited number of times and have limited our intruder to a single search.

Previous work [5] has showed how to adapt the standard CSP protocol model using ideas from data independence so that arbitrary-scale implementations can be proved secure. This is done by

[8] It is also important that the one-way pairwise version is one-way in that direction: if we take the opportunity to add a packet from B to A in the second message of the protocol, as well as the final digest, that message's authentication can be successfully attacked using reflection plus combinatorial search. This demonstrates that including unique node identifiers in the hashes of the key shares should be strongly recommended in *all* variants of the SHCBK protocol.

building finite-state implementations that have the ability to simulate arbitrary ones well enough to verify a security specification based on a very few trustworthy identities.

The finite-state models approximate the general implementation in the sense that they simulate all behaviours but potentially introduce extra ones and possibly false attacks. The crucial property that the intruder's deductive system must have in order for this approach to work is that it be *positive*: it must never rely on the distinctness of two values of the same type for a deduction to be made except, perhaps, where one or both are constants such as node names involved in the specification that do not change from protocol session to session.

If we restrict our attention to additional deductions introduced in Section 4 that model the search for weak values, there seems to be no reason why the data-independence framework should not apply. The silent values become constants of the sort referred to above, and then the additional deduction will preserve the necessary positive structure when added to the usual one representing strong encryption.

The issue of generalisation is more interesting for models that give the intruder the ability to search for values that solve equations. Such searches are typically relevant in protocols like SHCBK where the only use of weak values is in the comparison of a single set of them that are intended to be equal, as the last step of the protocol. Furthermore the touchstone of a successful attack is that two of these values are equal even though they are not constructed in the same way.

Under these conditions the only positive effect of a search by the intruder can be to create such an attack, and any such attack is enabled by the last search prior to the comparison that related to a constant involved in at least one of the two facts concerned.[9]

It should then follow that any attack would be found on any file modelling such a protocol would be found on a model restricted to a single search.

If we were allowed to use weak values for other things than comparison we might use them inside other constructions such as hashing and encryption (as done with passwords and short random strings in other types of protocol earlier in this paper). If the weak values used inside contexts were themselves the result of some construction, as in $hash(A, digest(hk, \langle A, B, N_a, N_b\rangle))$ then searching for equalities between these subterms and other weak values could enable attacks. This would greatly complicate the problem of finding a model for the general analysis of such protocols.

However under the restrictions discussed above this does not occur. Importantly, then, the acceptance of protocol messages at steps other than the final comparison will not be affected by whatever searches might have preceded them. Similarly, the interpretation of these messages by nodes at earlier steps will not be affected by searches. The only way in which a search can affect the progress of a protocol is in the final comparison. Being final, the success of such a comparison does not release any further information to the intruder.

It is therefore reasonable to conjecture that for such protocols the data independence techniques for protocol analysis will continue to work, in the sense that if there is an attack on any model of the protocol it will be found, augmented by giving the intruder a single search. A proviso on this is that the collapsing transformations used for data independence cannot remove an attack. There is no danger of this for the formulation of an attack where node B authenticates some constant originating from the intruder as being from A.

Further work is necessary to underpin the above arguments as well as to develop the CSP protocol models to implement them.

[9]The situation would be more complex in the modification suggested above to restrict the intruder to birthday-style attacks, since it is possible to perform such an attack by a search on two constants, one for each side of the equation, as in finding a collision in $digest(hk, hash(A, N_1)) = digest(hk, hash(B, N_2))$ by searching on N_1 and N_2.

It would, of course, be natural to build either our small implementation model or this generalisation into Casper.

One generalisation issue that the above does not address is the size of groups in those protocols that are run between more than two nodes. It seems to us that there is unlikely to be a one-size-fits-all approach to this problem. The case that seems most likely to be tractable is where protocols do not depend on the correct behaviour of all the participants, as in our analysis of the group version of SHCBK. Even there, however, we would need to allow for the collection of data from an arbitrary rather than fixed-size group. This will require some interesting abstraction.

9 Conclusions

We have successfully extended the well-tried CSP model of cryptographic protocol to encompass two different sorts of ways an intruder might exploit the presence of weak values and cryptographic constructs, and applied the resulting model to a number of protocols. Indeed, with the extension to semi-weak values contemplated in the previous section, we believe that our model is capable of coping with all of the protocols set out in [12], with the exception of the FlexiMac protocol that we have already discussed.

We have also argued that for core classes of protocol it should be possible to use the data independence based methods for proving general protocol implementations together with our new approach.

Using weak values automatically brings in a stochastic element to protocols: the relationship between the lengths of these values and the quality of security will generally be much closer to the front of the mind than in other types of protocol. In most cases the presence of a human participant applies downward pressure to these lengths, and so one wants to know just how long they really need to be to attain a given security level. One cannot just assume that the lengths will be well in excess of what it needed.

The main deficiency of the approach set out in this paper is that it does not give a numerical value for security: the highest probability of success of any intruder strategy. To get such a value, we would need to build a model in a probabilistic model checker or similar tool. This is an important topic for future work. It may also be possible to derive results about stochastic behaviour for restricted classes of protocols which show that a successful result from one of our yes-or-no checks guarantees that (for particular properties of the cryptography involved) the chance of a successful attack is no more than ϵ. The distinction between birthday-style and simple searches would be important, but as we have already seen we can distinguish these symbolically.

A question already posed in Section 4 is how our method of symbolically allowing for the deduction of preimages compares to Lowe's approach to guessing in [10], theoretically and pragmatically, with the latter meaning both efficiency and coverage in finding attacks.

References

[1] Please see: `http://www.prismmodelchecker.org/`.

[2] CSP files that illustrate the technique we discuss in this paper can be downloaded from: `http://www.cs.ox.ac.uk/publications/publication5265-abstract.html`

[3] D. Balfanz, D. Smetters, P. Stewart and H. Wong, Talking to strangers: Authentication in ad-hoc wireless networks, in: *Proceedings of the 9th Annual Symposium on Network and Distributed System Security (NDSS)*, 2002.

[4] B. Blanchet. *Automatic Verification of Correspondences for Security Protocols.* Journal of Computer Security, 17(4):363-434, July 2009.

[5] P.J. Broadfoot, G. Lowe, A.W. Roscoe. *Automating Data Independence.* ESORICS 2000: 175-190.

[6] J. Heather and S. Schneider. *A decision procedure for the existence of a rank function.* Journal of Computer Security 13(2): 317-344 (2005).

[7] A. Legay, A. Murawski, J. Ouaknine and J. Worrell. *On Automated Verification of Probabilistic Programs.* Proceedings of TACAS 2008, LNCS 4963, pp. 173-187.

[8] G. Lowe. *Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR.* TACAS 1996: 147-166.

[9] G. Lowe. *Casper: A Compiler for the Analysis of Security Protocols.* Journal of Computer Security 6(1-2): 53-84 (1998).

[10] G. Lowe. *Analysing Protocol Subject to Guessing Attacks.* Journal of Computer Security 12(1): 83-98 (2004).

[11] G. Lowe and A.W. Roscoe. *Using CSP to Detect Errors in the TMN Protocol.* IEEE Trans. Software Eng. 23(10): 659-669 (1997).

[12] 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).

[13] 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.

[14] 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.

[15] L.H. Nguyen and A.W. Roscoe. *On the construction of digest functions for manual authentication protocols.* Please see: `http://www.cs.ox.ac.uk/files/4130/digest.pdf`.

[16] 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.

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

[18] A.W. Roscoe. *Modelling and verifying key-exchange protocols using CSP and FDR.* CSFW 1995: 98-107

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

[20] A.W. Roscoe. *The theory and practice of concurrency.* Prentice Hall. 1998.

[21] A.W. Roscoe and M.H. Goldsmith. *The perfect spy for modelchecking cryptoprotocols.* In Proceedings of DIMACS workshop on the design and formal verification of cryptoprotocols. 1997.

[22] F. Javier Thayer, J.C. Herzog and J.D. Guttman. *Strand Spaces: Why is a Security Protocol Correct?* IEEE Symposium on Security and Privacy 1998: 160-171.

[23] 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.