

Reverse authentication in financial transactions and identity management

A.W. Roscoe, Chen Bangdao and L.H. Nguyen
Oxford University Department of Computer Science
E-mail: {Bill.Roscoe, Bangdao.Chen, Long.Nguyen}@cs.ox.ac.uk

September 7, 2011

Abstract

New families of protocol, based on communication over human-based side channels, permit secure pairing or group formation in ways such that no party has to prove its name. Rather, individuals are able to hook up devices in their possession to others that they can identify by context. We examine a model in which, to prove his or her identity to a party, the user first uses one of these “human-interactive security protocols” or HISPs to connect to it. Thus, when authenticating A to B , A first authenticates a channel she has to B : the reverse direction. This can be characterised as bootstrapping a secure connection using human trust. This provides new challenges to the formal modelling of trust and authentication.

1 Introduction

This is a paper about trust, security and identity management in the world of pervasive computing.

Over the past few years a number of what are sometimes termed “Human Interactive Security Protocols”, or HISPs, have been developed that permit one or more humans to bootstrap strong security between two or more devices based on the non-fakeable transmission of a minimal quantity of data between them to supplement a normal insecure communications medium. Because the humans know between which systems they have communicated this data (typically a few characters long and which we will refer to as a *check-string*) they know which systems are connected securely. There is an important difference between these protocols and those that bootstrap security from passwords, namely that the check-string does not have to be secret.

This class of protocols allows two or more parties who trust one another, or a single party who trusts one or more others, to bootstrap a secure network using no more than an ability to communicate a small number of bits over the human-based, non-fakeable channel denoted by \rightarrow_E . Another way of looking at them is that if the human(s) involved create an insecure channel between their devices, and already have an unfakable way of passing a small amount of information amongst them, then they can either turn the insecure channel into a secure one or discover the presence of an intruder who is trying to subvert it.

The best of these protocols, for example those of [13, 14, 20, 22, 23, 24, 25, 28, 33], enable these humans to be assured that there is no attack that allows an intruder to get the system into an insecure state (where the connections established are other than what the humans

believe) with probability meaningfully greater than 2^{-b} where b is the number of bits in the check-string. In addition, to have such a chance, the attacker will have $1 - 2^{-b}$ chance of his presence being revealed by the difference between the strings. In particular, these protocols prevent any combinatorial searching by the intruder improving its chance of success.

In many protocols of this type, parties or portable devices want to agree on the same data, such as their public keys, addresses and identities, because from the public keys they can easily create a strong private key that is used to encrypt or decrypt subsequent communication. In the absence of password and PKI, they will first exchange the public data over an insecure but high-bandwidth channel (e.g. WiFi or the Internet which are denoted by \rightarrow_N) and then display a short b -bit and non-secret digest of the protocol's run that the devices' human owners will manually compare to ensure that they have agreed on the same data, i.e. the latter uses human trust and interaction among humans to prevent fraud, including identity theft. This family of authentication protocols thus provide an easy way to bootstrap security that does not rely on the use of passwords, PIN numbers, shared private keys or any pre-existing security infrastructure such as PKI, which are known to be vulnerable to human misuse and misunderstanding in many circumstances in real life. For example, passwords are frequently chosen from a limited range which makes them vulnerable to guessing. PKIs are not straightforward for many human users, and thus it might be more convenient for humans to create a secure communication in online payments, telephony or e-healthcare from existing human trust and interactions as required in this new type of authentication technology. In addition to many types of applications, HISPs can be used in a wide variety of ways, in contexts both where all the devices are co-located and where they are not, and where the authentication is provided to all devices or asymmetrically to one, because only that device's user has observed the equality required of the check-strings. Similarly they can be used in convenient consumer devices or as part of the security process in a more elaborate type of system.

This is a new approach to security and requires novel approaches at multiple levels. In this paper, we will first describe a HISP which is based on the SHCBK protocol of the authors [22, 23] in Section 2 and then elaborate on the advantages of our approach in payment and ID management in Sections 3–5. In the second half of this paper, we will provide an overview of some of issues on automated verification techniques in Section 6, a new cryptographic primitive called a digest function in Section 7, and implementation in Section 8 that this new class of authentication technology creates. We summarise our contributions in Section 9.

Acknowledgements

We are grateful to Ronald Kainda and Ivan Flechais for their work with Roscoe on the human factors of HISPs and to Toby Smyth for his work on the formal verification of these protocols. A number of researchers from the banking industry have helped us to understand what is required there and enabled us to understand the real-life problems that protocols for financial transactions need to solve.

Parts of the work reported in this paper have benefited from funding from QinetiQ and the US Office of Naval Research.

2 Example protocol

The following protocol was designed and implemented by the authors as the first phase of a larger one designed to perform a financial transaction securely. It is closely based on the

SHCBK protocol of [22, 23]. Because of its intended application we call the two parties C (customer) and M (merchant). Before the protocol is run these two parties have no shared knowledge that helps them achieve security, except, naturally, the ability to run the protocol itself.

In order to run the protocol each party must create two values of sufficient strength to achieve the cryptographic goals they have:

- Each party creates a *hash*, or *digest* key: we call these hk_C and hk_M . These are needed to randomise the final check-string and we assume these are in the range 160-511 bits¹.
- C creates a session key k whose role is as set out above. This would normally be in the range 120–160 bits, but it could be increased to 512 bits (input width of the basic hash function) without penalty.
- M either creates freshly, or re-uses, an asymmetric key pair (pk, sk) . There is no need for the “public” key pk to be certified. The length of these keys will depend on the desired level of security², the amount of available computing power, and the cryptosystem in use.

The protocol also involves a standard cryptographic hash function which possesses 3 main properties [19]: collision resistance, 2nd preimage resistance and inversion resistance. It depends heavily on the following property of such a function:

If a party A has knowledge of $hash(V)$ for some value V , then while A cannot constructively compute V other than with infinitesimal probability, it can always check in future whether a value alleged to be V actually is V .

In this state A is *committed* to V but *without knowledge* of V . In the first pair of steps of the protocol, C and M both commit each other without knowledge to values. The only one of the four parameters hk_C , hk_M , pk and k communicated openly is M ’s public key pk :

1. $C \rightarrow_N M : hash(0 : hk_C), hash(k)$
2. $M \rightarrow_N C : hash(1 : hk_M), pk$

When these messages have been received, both parties are committed to values of all four parameters, but each lacks some knowledge. Thanks to the communication channel being insecure, they have no reason to believe that they are committed to the *same* values of the parameters – but of course they hope they are! Importantly, no intruder can know all four of the original (as opposed to hashed) values as created by the appropriate one of C and M .

The tags 0 : and 1 : are added to the hash keys hk_C and hk_M to ensure that the contents of these hashes can be distinguished as coming from a customer or merchant. This avoids the intruder reflecting hk_C back to C as a supposed hk_M in a way that C would accept, i.e. a reflexive attack. In Section 6, we will use automated verification via CSP and FDR to demonstrate that this protocol is vulnerable to such a reflexive attack when the hash keys are not hashed with the tags.

¹The upper bound takes account of the common hash block size of 512 and the extra initial bit inserted by the protocol.

²The key certainly needs to be strong enough so that there is no realistic chance of it being broken during the life of the session being established. Further strength is required to ensure that the contents of that session remain secret after it ends.

Even if the intruder has participated in the protocol and impersonated one or both of the parties, it does not know the complete set of parameter values to which either C or M is committed. This is because no-one except C knows its value of hk_C , and similarly for M and hk_M .

The protocol now proceeds:

3. $C \longrightarrow_N M : hk_C, \{k\}_{pk}$
4. $M \longrightarrow_N C : hk_M$

The second part of Message 3 is to tell C the actual value of the session key,³ which is now checked against the hash. In fact this transmission can be delayed until after the check-string comparison if the computation of the public-key encryption $\{k\}_{pk}$ is time consuming on a low-powered customer device.⁴

It is the transmission of the unencrypted keys hk_C and hk_M at this stage that represents the core of the protocol. Firstly, of course, the participants must check that these are the same values that were represented in Messages 1 and 2. If not, the run is abandoned. Secondly, they (and anyone else who has been listening in) can compute a value for

$$digest(hk_C \oplus hk_M, (pk, hash(k)))$$

where \oplus is bit-wise exclusive or and (X, Y) is an ordered pair. The protocol completes successfully if C (or C and M) are convinced that their two versions of the value – the check-string of this protocol – are equal: in becoming convinced they must not use a channel which can be “spoofed” by an intruder. Typically one will read their value to the other, or C will read M ’s value directly and compare it with her own. Whichever of them knows that the two values are equal can conclude that the link is authenticated. Typically this is either C or both of them.

Naturally, if the protocol has proceeded uninterfered with, C ’s and M ’s values will be equal. If, however, an intruder has imposed his own values on the receivers of Messages 1–4, C and M will not agree on all four parameters. For security, what is important is that they agree on pk and $hash(k)$, so we will concentrate on what happens if the intruder interferes with these.

What we are concerned about is the chance that the digests agree when these two values do not.

The digest function [21, 22, 23] is designed so that, as hk varies, the probability that $digest(hk, X) = digest(hk, Y)$ for $X \neq Y$ is less than ϵ , where typically ϵ is very close to 2^{-b} for b the number of bits in the output of $digest$. It must also have the property that for any fixed value d , the chance that $digest(hk, X) = d$ as hk varies is less than ϵ also. We will describe some results on the construction, theory and computational cost of these functions in Section 7.

The following is an argument for why this protocol is secure. The reader can find similar arguments in [22, 23]. In Section 6 we will discuss the problem of using model checkers to verify this style of protocol formally and the results that have been obtained.

The intruder can easily convince either or both of C and M that the other’s hk is not the one it should be. The only result of this activity, if done, is that as far as can be determined

³Formal analysis later, in Section 6, will show us that there are advantages in replacing this encryption by $\{k, hash(hk_M)\}_{pk}$.

⁴This did not prove to be an issue in our implementations: see Section 8.

effectively by an intruder, C 's and M 's views of hk^* are independent uniformly distributed random variables: this is because of the following property of the bit-wise XOR used to construct it.

If X and Y are independent random variables and X is uniformly distributed, then $X \oplus Y$ is uniformly distributed whatever the distribution of Y .

Honest parties C and M will have chosen hk_C and hk_M uniformly at random, and independently of the other values in the protocol⁵.

This means that the chance of C 's and M 's digests being equal is $\leq \epsilon$ whether their views of pk and $hash(k)$ are the same or not. We can conclude that it is not a good idea for the intruder to get C and M to disagree about hk_C and hk_M . So henceforth we will assume they agree on these values: necessarily the ones they picked for themselves.

At the point just before Message 3 is sent, C knows that only she knows the randomly chosen hk_C . Therefore (whatever value was picked by M for hk_M), she has effectively randomised the final digest in a way that no-one else has knowledge of. M knows the same about his injection of hk_M . Each can therefore reason separately that the chance of agreement between the two instances of $digest(hk_C \oplus hk_M, (pk, hash(k)))$ if their values of pk and $hash(k)$ are different is less than ϵ .

The crucial point is that each of pk and $hash(k)$ were known to both parties before the function that would be applied to the pair $(pk, hash(k))$ was known to anyone: indeed so far as any party is concerned, the value $hk_C \oplus hk_M$ which determines that function remains uniformly distributed over all possible values until C and M have revealed hk_C and hk_M . It is this that means that there is no point in the intruder doing any combinatorial search against this function: by the time it is known, it is impossible to change the minds of C and M about what to apply it to.

Given that C and M do agree on $(pk, hash(k))$ it is also true that they agree on k , thanks to the assumed properties of $hash()$. When C sent $\{k\}_{pk}$ she did not know who could understand it, but when she knows that pk is the value that M sent her she then knows that no-one other than herself and M know k . Similarly C , on receiving $\{k\}_{pk}$, C does not know who it came from, but once he checks that the hash of the key received this way corresponds to the value $hash(k)$ he and M agree to, he knows that he and C agree to M and that no-one else knows this value.

The above describes the principles behind this protocol. There are a number of others that operate in ways that, from the perspective of the human users, are either very similar or identical [11, 13, 14, 33], and see [24] for an extensive survey. A number of them can be adapted for group use. SHCBK, from which ours is adapted, was in fact *originally* designed as a group protocol.

The whole purpose of this protocol is to establish k as a secret key shared between C and M and known to no other party, on the assumption that both are trustworthy, if the digests agree. In our intended application k will be used to secure and authenticate communications between the two: only C or M could have created any data encrypted (under a symmetric algorithm such as AES) using k and only they can understand it. So, in particular, C knows that the payment details sent to and received under k are with the intended party.

We will gain further insight into the role and properties of k in Section 6, as well as showing how to modify the protocol to make it clearer.

⁵The fact that hk_C is independent of the value of hk_M it accepts depends on the tagging of these values by 0 and 1.

3 The relationship between trust and authentication

In many cases such as a financial transaction over the Internet or with a vending machine, or someone seeking to prove his identity and gain access to some service via a machine, there will only be one human present to perform the check of the equality of the digests or similar value used by other HISPs.

For high integrity applications involving a potentially complacent human in this way, there is a strong argument for having the human transfer a value manually rather than simply check that two displayed values are equal. So in fact the device actually performs the comparison – between the one its user has copied from M and the one it has computed itself. There are still some interesting variants possible on this. In the following, we will split the entity C , representing the combination of a human customer Alice and her security device SD , into these two parts.

CA Most obviously and probably easiest in many applications: Alice reads a value from M and types it into her own device SD .

CS As a variant on this: when SD says that the two values agree, Alice presses a button on M to signal this agreement.

MA Alice reads the value from her SD and types it into M . In this case it is virtually certain that M will signal to the human whether the two values agree. This information may or may not be passed explicitly by Alice on to SD . (iii) will be the case where it is not.

MS This is the case like (iii) but where this information is transmitted by Alice to SD .

The pairs CA and MA, and CS and MS are clearly mirror images of each other: the first letter tells us who does the comparison in a method, and the second tells us whether the resulting authentication is asymmetric or symmetric. In CA and MA only a one-directional *empirical* channel is used between SD and M , in opposite directions. In CS and MS, both sides are assured of equality provided our human is trustworthy and reliable.

Note that in cases CA and MA one or other device proceeds without *knowing* that the check-strings actually agreed. Nevertheless these two cases are potentially useful: for example CA can be used when everything confidential and of value that passes through the transaction moves in the direction from SD to M .

The most obvious case of this is Alice using SD to pay M for some goods and services: M wants to get paid but does not care that much who pays him, whereas Alice only wants to pay the merchant to which she has an obligation to pay.

Notice that this reasoning does not apply when the merchant is passing value to a customer. Imagine that the merchant wants to pass value to the customer who is standing at a particular till, is on a particular phone call etc. If the protocol is run in mode MA it gains the assurance that it is this person's device that is connecting to it, since only she was in a position to make the empirical communication. In practice, however, this situation might well require the customer to pass (e.g. ID) information to the merchant that she would not want to give to *anyone*, and it would be better for Alice not to need to learn a very different protocol for a relatively unusual case. Therefore a protocol giving a symmetric outcome would probably be used here, in particular CS with a warning to Alice about the consequences of pushing the final button incorrectly.

One way of more-or-less ensuring that she does behave correctly and not push the button without knowing that the digests agree is to have both sides compute a bit from the protocol

parameters, and use it to have M only tell Alice which of *two* buttons to press once it knows the digests/check-strings agree. The essential thing here is making the customer look at the device that has been able to check equality before confirming anything to the other one.

We might note that it will almost always be Alice who identifies that M is what she wants to connect SD to, and that the connection process itself gives neither party any proof of the identity of the other. Therefore, at the end of the HISP run:

- Unless MA is used, Alice knows that C is connected to M , as identified by being at the other end of the empirical channel.
- Unless CA has been used, M “knows” that the party it is connected to is the one at the other end of its empirical channel.
- Both know they have a shared secret symmetric key with the other, that can be used to secure and authenticate communication between them in a subsequent session.

4 Case study: supporting a financial transaction

The connection above has very little in common with the way that most personal financial transactions are performed, at least those involving banks⁶. For the current methods for doing these, whether manual (e.g. typing details from a credit card into an `https` site) or electronic (e.g., logging in to Internet banking; using Chip-and-PIN terminals at point of sale) are designed simply to authenticate the payer (Alice and/or her credit card) to the payee and/or bank. Typically also, traditional methods of payment give the payee a great deal of sensitive information about the payer: note that anyone who has been paid on-line with a particular card has the information he (or anyone to whom it is unwittingly compromised) to pay for goods with that same credit card; and that nothing proves to Alice that the Chip-and-PIN terminal in her hands will not clone her card and remember her PIN.

What the above protocol does is allow Alice to connect her SD to the particular merchant she has been shopping at, whether in person, on-line, or on the telephone. Clearly this authentication is in the reverse direction to the usual sort as described in the preceding paragraph. That explains the title of this paper: it can be viewed as *reverse authentication*.

What we have allowed Alice to do is to create a secure electronic connection with the merchant that she wants to pay, and furthermore where she is assured that the connection has been made within the context of the *particular* transaction for which she is willing to pay.

This electronic connection will allow a great deal more information to pass between her device (card/SD) than is otherwise possible unless she puts her card into the hands of the payee, or at all on-line.

What the reverse authentication achieves, in summary, is not to *replace* the usual ID check on Alice and her device, but to make it potentially more thorough, particularly on-line, easier for Alice (because in most cases she will have to do less), and to remove the danger of Alice’s secrets getting compromised. We will demonstrate this below.

We believe that beginning a financial transaction with the HISP authenticating merchant to customer makes sense in all of the following cases.

⁶As we will see, good old fashioned cash transactions resemble our connection.

- (A) Electronic cash: the customer has some device with her that contains value, and she wishes to transfer some of that value to the merchant. It might well be the case that she wishes to do so anonymously.
- (B) Credit card or cheque: the customer wishes to give the merchant the right to take some sum of money from her account. Note that, although the mechanisms are rather different, both conventional credit card transactions and paper cheques have this logical effect.
- (C) Electronic banking: the customer wishes to give her bank a direct instruction to pay the money into the merchant's account. The logical difference with (B) is that the bank must be involved directly, and the merchant never holds a token that is good for money.

This first dimension influences the way the payment proceeds after a secure link has been established, and how identity issues arise. (A) is different from the others because our customer will not have to prove her identity (Alice) to anyone, while in the other two it is desirable⁷ that she (separately from her device) proves that she is entitled to use the account by the entry of a secret PIN or biometrics.

In each of these we imagine a variety of payment situations, which influence how the authenticated connection is made.

- (i) The customer is sitting at a desk and shopping on-line. Here we assume that the customer and the banking system do not trust the PC except possibly through an `https` windows displayed on browsers. [This is not to say that this last mechanism is 100% secure, but since e-commerce and digital banking rely on it currently, it seems reasonable that we can also provided we do not increase the risks inherent from using it in present methods⁸.]
- (ii) The customer is trying to pay the merchant in person: in present technology she would hand over cash or credit card, or place her card in some reader presented by the merchant. Here, the merchant might be a machine, or might be a manned till.
- (iii) The customer is shopping over the phone.

In case (C) (mobile banking) we do not concern ourselves with how the secure connection between phone and bank is made, since we can reasonably assume that there is a long-term key which achieves this. In all cases we assume that a HISP is to be used to connect the phone to the merchant that is to be paid, thereby proving to Alice that she is paying the correct entity within the correct transaction.

Except in the case where the the paying device SD is the same telephone over which the transaction is being conducted, we need to get some connection between SD and merchant established to that it can be authenticated with a HISP. In technology used for everyday payments, both of these phases need to be very easy. One advantage of using a HISP is that the customer's view of the second phase can be the same in every case. The following sets out a few options for making the initial insecure connection in the on-line and point-of-sale cases.

⁷We note that in some present credit card transactions, especially on-line ones, she does not have to do this.

⁸In fact we argue that our methods provide a higher degree of security than the traditional use of `https` sites, since we are only relying on the communication through it being authenticated, not secret. Thus neither screen-shot grabbing nor key-sniffing would benefit an attacker.

In the on-line case there are two main options for this connection: the first of these is using the home PC on which the shopping is being done as a link between SD and merchant. The link could then be made by wire (e.g. USB), wireless (e.g. WiFi or Bluetooth) or infrared. None of these is technologically difficult, and the session on the browser can instruct the PC about where to route the communications.

The second is using telephony to make the connection: this may be the only option when Alice is forbidden to connect any personal device to the PC. The only problem then is giving one or other side of the connection (phone=SD or merchant) the information required to connect telephonically to the other. This will be the combination of a telephone number and a (probably one-time) token that identifies the particular transaction. The merchant's number can be transferred to the phone from the PC by (e.g.) Bluetooth, but of course this would fall foul of the no-connection rule if this applied. The user's number can be pre-loaded into the browser (and there are strong arguments for this being a separate number from the one used for ordinary phone functions), and this sent together with a one-time token to the merchant when a button on the payment site is pressed.

In the point-of-sale case, the same two options (local and telephonic) connection apply. A literally wired local connection is unlikely, but it would be possible to place a phone into a special cradle. In that case it is probably not necessary to use a HISP, since the phone is obviously connected to the merchant. Similarly, if a connection is bootstrapped from a physical connection that Alice can *see*, this is still probably not necessary, and the same may apply for low-value transactions if it is bootstrapped from very short range radio as used for example, in Oyster cards [4], provided this generates a session key. A HISP will provide additional assurance in this last case for high-value transactions. Other options include Bluetooth connection (where it may be necessary for Alice to select which till she is at) or telephony (where the number can be transferred using any of the methods described above, or perhaps via scanning of a bar-code displayed on the phone).

In all payment methods, we assume the first action after the secure session is established would be for the merchant to send the SD details of the transaction it wishes to be paid for plus secondary security information such as its name and logo. If Alice agrees to the payment, she will either press a button or enter the personal information (e.g. PIN or biometric) needed to confirm her presence. Whatever payment token is then sent by her SD will then contain the secondary security information so that a fake merchant who has "borrowed" these should not be able to obtain payment from them.

An electronic cash payment would simply follow the appropriate protocol over the secured session.

For a credit card transaction we either have greatly increased the communication possibilities between SD and merchant or (particularly when Chip-and-PIN terminals are replaced) ensured that there is much less availability of customer information to merchant. In either of these cases it makes sense to replace present payment methods by the SD giving the merchant an e-cheque containing

- Payee, payer, amount, credit card details and time-stamp, as on a conventional cheque.
- Transaction ID.
- Any secondary security information about the payee that Alice has confirmed. *The bank will confirm that this ties up with the payee.*
- Evidence that Alice has correctly proved her own identity. This might be either the actual information (e.g. PIN) she has input, or evidence both that the SD/card has

confirmed this information (noting that at present PINs are typically confirmed by a credit card and not transmitted) and that the SD/card itself is genuine and behaving properly.

This would be encrypted under a key that merchants cannot understand (e.g. a symmetric key specific to this SD/Card or the public key of the banking system) and sent to the merchant to be forwarded and authorised by the banking system.

Perhaps the most attractive scenarios for using HISPs for payment comes in the context of *mobile banking* (i.e. on-line banking on a mobile phone). Systems implementing this with limited functionality are rapidly being developed by banks and rolled out to customers, but none that we are aware of allow the user to make a payment a general point-of-sale or on-line merchant. The deficiency can be remedied once the phone is securely connected by HISP to the merchant. For then M sends details of the transaction for Alice to confirm, plus bank account details to which the money is to be paid. When C confirms and gives whatever authorisation code is required by her bank, the on-line banking session automatically generates a transfer to M 's bank, and an unforgeable certificate that this has occurred is sent by C 's bank to M via C . The value of the HISP here is that it ensures that the bank account details really come from M .

It is worth noting that the total effort that Alice has to make in running a HISP and confirming the transaction on her SD is substantially less than is required of her in conventional on-line purchases using credit cards, whether these are performed by entering card details onto a web-site or by entering her PIN into a secondary device provided by the card issuer and then copying a one-time authentication code into the web site. (Devices such as these are, of course designed to help Alice prove her identity – they do not help Alice to create an authenticated connection to the merchant.)

5 You, me, us and anonymous authentication

One of the strangest qualities that HISPs have is that they provide essentially *anonymous authentication*: they can provide security in contexts where the parties being connected do not know each others' names. We can imagine this being extremely useful in some applications such as electronic cash where a payee and payer might want to be assured that it really is them who are connected for some transaction without actually revealing their names to each other. The curious thing is that this situation is completely familiar when carrying out transactions with good, old fashioned, cash. What HISPs can do is to enable human(s) to connect devices that they trust because of their context without knowing their names.

In many situations the personal pronoun *you*, whether singular or plural, typically indicates that the speaker is addressing a collection of people that both the speaker and they understand precisely (i.e., what collection of people is being addressed), without carrying any implication that the participants know each other's names. The sort of authentication given to a single individual by running a HISP with such a collection might thus be termed *you*-authentication, while the sort achieved by a group who are all aware of the final agreement might be termed *us*-authentication for similar reasons.

Thus these protocols allow humans to build networks of secure communications amongst the devices that they trust, reflecting the same intuitions that guide their everyday life. For example, each of us develops a good sense of when to hand over money, or our credit card, to pay for goods and services, and who to hand these things to. This sense is largely unrelated to our knowing the identity of the person or other entity we are paying. Indeed, in many

cases, one either does not know this identity – how many times do you look carefully at the name of the filling station where you buy petrol? –or this name is largely irrelevant – how would it change your behaviour if you had noticed this name? What matters is context: the payer believes that the payee either has or will supply whatever it is that she is paying for, and accepts the contract to supply these in return for money. What our protocols achieve in this context is that they allow the payer to connect her account to the payee over an insecure network, so that the link becomes secure and she knows that the communication partner is the same entity that her intuition would have told her to pay by more conventional means.

Similar situations will arise whenever a party Alice carries some piece of paper, a card etc that either carries some less-than-public information (often about her), or which enables something of value to be obtained or transferred. Examples are a passport, driving licence, ID card, medical details (whether on a record card or derived from some sensor attached to Alice), company pass, tickets for travel etc, and membership cards. If these things are held on some electronic device, then Alice will want to know they are connected to the entity that she trusts with this information (the same one to which she would be happy to hand over the same details on paper). HISPs appear to be the ideal vehicle for this.

The role of her handing over this information to Bob (analogous to M) may simply be to prove her identity to him. She may be very reluctant to let anyone else see this information because of the dangers of her identity being stolen.

It is interesting to reflect at this point that she may well be interested in proving her identity to Bob without giving him the wherewithal to steal it, or “mis-lay” her information by not storing it carefully enough.

There is a potential advantage to Alice in the “card” or similar remaining in her possession on her own electronic device rather than being handed over to the recipient. This is that she can limit the transfer of information or value to what are necessary for the particular transaction, whereas a physical card or piece of paper might well be copied either as part of the normal procedures of Bob, or if he were less trustworthy than Alice believes. Even the first case is dangerous to Alice unless Bob protects her information better than has frequently been the case. Our electronic payment scenario illustrates this perfectly: if Alice hands Bob her credit card, she is giving him the means to charge unlimited transactions to her account. On the other hand, if she merely connects her account to him, she can hand him an e-cheque, usable one-time only for a stated amount of money. The latter would not be practical without an electronic connection, because it is too large a piece of data to transfer by hand.

Thus the approach we have devised for payments clearly provides a useful model to consider for pure identity verification, with the financial component removed. Just as in the case of a payment, the particular entity Bob with which Alice connects may only be *semi-trusted* by her. In the payment case, she wants to avoid disclosing long-term secrets to Bob and only give him the exact amount of money she is trading with him. In other cases she may only want to give the exact amount of information required by Bob’s apparent function, and may wish (as in the payment) to delegate the confirmation of her identity to a third party whom she and a legitimate Bob *fully* trust. As in the e-cheque case, the third party may sit behind Bob, or, as in the e-banking case, it may sit behind Alice. Depending on the context, there may be also be analogues of the secondary security information used in payments.

6 Formal modelling: challenges and insights

The science of cryptographic protocols has been transformed by the development of tools that attempt to verify that they achieve their security goals. The behaviour of such protocols can be remarkably hard for humans to understand in the presence of intruders who try to subvert them, and these tools frequently give remarkable insights – positive or negative – into their behaviour. In this section we discuss the application of such tools to variants on our protocol.

Unfortunately, established analysis methods only give a very incomplete picture of HISP protocols. Since it can prove difficult to motivate human users to put in a lot of effort in the name of security, HISPs present the unusual challenge to protocol verifiers that we cannot just assume that all cryptography is so strong as to make combinatorial analysis, cryptanalysis, etc., pointless. Protocol analysers (whether human or automated) usually make this assumption, so whereas the theory and tools for verification under this *perfect cryptography* assumption are now very powerful and sophisticated, they simply do not apply in our case. So the best that established tools can do is to prove security in an environment where the attacker does not attempt to exploit the opportunities for combinatorial search offered by the use of short comparison strings.

For a convincing analysis we need to extend the powers of the attacker inside a tool so that it can exploit these opportunities.

We have identified two distinct approaches to this problem. The first is to attempt to capture what is weak about the use of short digests symbolically and discretely, so that we can adapt existing model-checking or theorem-proving approaches to protocol verification. Such approaches can still be expected to give a yes or no answer to the question “is protocol X secure”. The other is to try to quantify security, for example by calculating the probability with which an optimal strategy by an attacker can lead to a breach of security.

At the time of writing we have successfully incorporated methods taking the first of these approaches into the CSP style of protocol as described in [3, 27, 30]. This work is reported in a new paper [29], and ideas from that paper relevant to the present paper are summarised below, together with results obtained from analysing variants on the protocol from Section 2 which we recall below: \rightarrow_N and \rightarrow_E denote normal insecure but high-bandwidth and empirical channels respectively.

Pairwise key-establishment version of SHCBK protocol, [23]	
1.	$C \rightarrow_N M : \text{hash}(0 : hk_C), \text{hash}(k)$
2.	$M \rightarrow_N C : \text{hash}(1 : hk_M), pk$
3.	$C \rightarrow_N M : hk_C, \{k\}_{pk}$
4.	$M \rightarrow_N C : hk_M$
5.	$M \leftarrow_E C : \text{digest}(hk_C \oplus hk_M, (pk, \text{hash}(k)))$

6.1 Modelling digest collisions

The weakness of digests used in HISP compared to traditional hashes can be captured by building in the assumption that, given a digest value d and an opportunity to modify either of the values hk or X while knowing the other, it is possible to find such values with $\text{digest}(hk, X) = d$. In other words we can ensure that the pair (hk, X) is a preimage of d . The approach of a real attacker to finding these values would be to search through a range of possibilities for hk and/or X ; in our formal model we need to model this symbolically.

Note that this assumes greater weakness than merely assuming the efficacy of the birthday attack, which would be to remove the above assumption and replace it by the possibility, given the opportunity to modify (at least) one each of (hk, X) and (hk', X') , to ensure that $digest(hk, X) = digest(hk', X')$. Clearly a protocol proved secure under the first assumption is also secure under the second, and given the possibility of using digests so short that attacks of the first sort have a good chance of succeeding, we have concentrated on that.

We want to emphasise that in this approach the intruder is modelled to look for different strong values (e.g. long cryptographic key of say 160 bits or large data) that produce a weak value when placed in a context. We therefore formalise this induction rule as follows

Collision induction rule [29]: For every weak value w and every weak-valued context $C[\cdot]$ that the intruder knows, whose value depends on its argument, the intruder can search for v such that $C[v] = w$.

This is an ability that is very different from the usual sort given to the intruder – even guessing weak passwords [15] – because rather than allowing it to learn new combinations based on existing constants, we want it to introduce a new one with a special property.

What we have observed in [29] is that the sort of HISP of Section 2 and many of its variants described at [24] 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.
- A set of weak values are compared at the end of the protocol – two in the case of a pairwise protocol. An attack is realised if two trustworthy participants have equal values that are computed from different inputs.

Using the observations, our CSP model takes a snapshot of its memory whenever the intruder invents a fresh value, and the model asks each time two weak values are compared whether the intruder *could* have performed a search to force them to be equal using the most recent fresh value created by the intruder.

6.2 The security of agreement

Observe that the $\{k\}_{pk}$ component of Message 3 plays no role in the process of the two parties agreeing on the digest or the values of pk and k (as $hash(k)$), but is necessary to give M knowledge of k .

We therefore chose to analyse two versions of our protocol, one with that component missing, designed to check that M and C always agree on the values of k and pk , and a second which exploits it in an extremely simple model of payment: sending a payment token from C to M , abstracting away details of the transaction, date and amount, from C to M and encrypted under the session key k : $\{pay(C, M)\}_k$.

No attack was found on the first, demonstrating that despite the intruder’s ability to perform searches, agreement on the final digest implies that the participating customer and merchant agree on the values of k and pk .⁹

⁹There are two important caveats we must make to this claim. The first is that our checks were carried out on models in which we only examine a single pair of trustworthy nodes able to run the protocol up to twice each against an intruder with a single identity to use. There is good reason to believe that that this would find any attack, but we have yet to integrate our model of combinatorial search into the CSP techniques [30] that prove general versions of protocols under strong encryption. The second caveat is that nothing can prevent the possibility of the intruder making a single lucky guess that results in a digest collision with a probability of about 2^{-b} where b is the width of the digest.

In order to check the accuracy of our CSP model, we ran it again but on a weakened version of the above protocol where the hash keys hk_M and hk_C are hashed alone in the first two messages instead with tags "1:" and "0:". The protocol therefore becomes

Pairwise version of the weakened SHCBK protocol, [22]	
1.	$C \longrightarrow_N M : hash(hk_C), hash(k)$
2.	$M \longrightarrow_N C : hash(hk_M), pk$
3.	$C \longrightarrow_N M : hk_C, \{k\}_{pk}$
4.	$M \longrightarrow_N C : hk_M$
5.	$M \longleftarrow_E C : digest(hk_C \oplus hk_M, (pk, hash(k)))$

This is similar the original version of SHCBK that we presented at a workshop [22]. The tags 0 and 1 are based on the introduction of node names inside the corresponding hashes in the definitive presentation of SHCBK in [23], which were introduced because we were aware that the use of \oplus in combining these hash keys could allow the intruder to reflect hk_C back to C as its own, so that the randomness contributed by hk_C to the computation of the manually compared weak value or short digest is eliminated.

Our CSP checking model indeed discovered an attack on the weakened version of SHCBK, both in its group version and in the binary model above. In this attack, the intruder plays the role of the man-in-the-middle: except for the human comparison of the short digest, the intruder poses as M to interact with C and poses as C to interact with M in two parallel protocol runs. As seen in the following description, at the point when the intruder discovers hk_M in Message 4', it will be able to search for a different pk' such that

$$digest(hk_M, (pk, hash(k'))) = digest(0, (pk', hash(k)))$$

Observe that at the point it delivers Message 1' with its own choice of session key, the intruder already knows that it can persuade C to use hash key $0 = hk_C \oplus hk_C$.

An attack on the pairwise version of the weakened SHCBK protocol	
1.	$C \longrightarrow_N I(M) : hash(hk_C), hash(k)$
1'.	$I(C) \longrightarrow_N M : hash(0), hash(k')$
2'.	$M \longrightarrow_N I(C) : hash(hk_M), pk$
3'.	$I(C) \longrightarrow_N M : 0, \{k'\}_{pk}$
4'.	$M \longrightarrow_N I(C) : hk_M$
2.	$I(M) \longrightarrow_N C : hash(hk_C), pk'$
3.	$C \longrightarrow_N I(M) : hk_C, \{k\}_{pk'}$
4.	$I(M) \longrightarrow_N C : hk_C$
5.	$M \longrightarrow_E C : digest(hk_M \oplus 0, (pk, hash(k')))$
	$C \longrightarrow_E M : digest(hk_C \oplus hk_C, (pk', hash(k)))$

The consequence of this attack is clear: the intruder discovers the key k that C wants to send to M as well as fools M into believing that a different key k' comes from C .

6.3 Analysing the full protocol and the role of k

We now turn our attention to the full protocol with the addition of the final payment message discussed above. The condition for the sending of this message is that the customer C observes

equal digests. In many potential applications (e.g. online and vending machines) it will only be the customer that observes this since there will be no human presence on the part of the merchant. We therefore placed no digest agreement condition on the merchant *accepting* a payment message under the session key of its current protocol session. This is the same as the agreement mode CA discussed in Section 3. The protocol therefore becomes

Payment version of pairwise SHCBK protocol, [23]	
1.	$C \rightarrow_N M : hash(0 : hk_C), hash(k)$
2.	$M \rightarrow_N C : hash(1 : hk_M), pk$
3.	$C \rightarrow_N M : hk_C, \{k\}_{pk}$
4.	$M \rightarrow_N C : hk_M$
5.	$M \rightarrow_E C : digest(hk_C \oplus hk_M, (pk, hash(k)))$
6.	$C \rightarrow_N M : \{pay(C, M)\}_k$

Our model also had the merchant use the same public key for all its protocol sessions: while there is every reason to expect the session key k to be fresh every time, there is no immediately obvious reason why the merchant should have to perform the computationally difficult task of generating a fresh asymmetric key pair every time. We are, after all, used to public keys having a long lifetime.

Our model was designed to detect three different sorts of behaviour that one might regard as erroneous.

- i. The intruder learning the value of the token that a particular customer C uses to pay a merchant M .
- ii. M accepting a payment which he thinks is from C but was actually sent by someone else. [So someone else settles C 's debt.]
- iii. M accepting a payment which he thinks is from a third party, but is actually the token C uses to pay him.

We will discuss the results of these in turn. The first specification succeeds (i.e the situation described does not occur), though of course if the hash tags are removed the reflection attack is available on the agreement protocol and it fails.

The second specification fails: it is possible for anyone to pay M as C . This reflects the asymmetry of payment: C clearly has an interest in making sure that she does not pay anyone else's debts, but very little interest in making sure that no-one else pays hers. If the merchant insisted on digest agreement prior to accepting payment it could ensure that the payment has come from the party it has run the protocol with. This reflects our discussion in Section 3 about the distinction between CA and CS.

Of course most payment methods are a lot more complex than simply sending a single token, and as we have discussed most require the payer to prove their identity to at least the bank or entity holding their account to establish that they are allowed to authorise the payment. Such a protocol might or might not reveal this identity to the merchant, but of course unless the merchant knows the name of the customer standing in front of him, or he knows that the digests are agreed, there is nothing to tie this identity to the person performing the protocol.

The third specification also fails because the intruder record a session between C and M where he hears the message components $hash(k)$, $\{k\}_{pk}$ and $\{pay(C, M)\}_k$ (where pk is the

public key M uses in every session) and then, when himself paying M , can substitute these components into Messages 1 and 3 of the protocol and then replay the last as the payment.

In practice this attack would not work in payment applications, because the payment message would be a lot more complex and would almost certainly include a serial number for this transaction that has been communicated to the payer by M .¹⁰ The intruder could not substitute this into the payment message because he never gets to know k . Nevertheless it is not really satisfactory that an intruder can apparently agree the same key with M that C did.

It follows that it would be better to modify the protocol to avoid this replay attack. A simple way of doing this is to include the merchant’s hashed key share $hash(hk_M)$ in the encrypted component of Message 3, so this message becomes¹¹

$$3. C \longrightarrow_N M : hk_C, \{k, hash(1 : hk_M)\}_{pk}$$

Note that C cannot substitute $hash(hk_M)$ by hk_M because she does not know this value. In effect this proves to M that whoever constructed this message knew both k and the session specific $hash(hk_M)$ when the encryption was created: it binds this component to the specific protocol session that M is running.

This modification means that specification (iii) now holds. In some sense the inclusion of $hash(1 : hk_M)$ in Message 3 is similar to including the hash tags: it makes it more explicit what a message component is for. In the case of the hash tags we make a distinction between those generated by customer and merchant. In the present case we tie the encrypted component of Message 3 explicitly to the particular session the merchant is running, preventing a replay from earlier sessions.

Explicitness is a well-established [6] principle in the design of cryptographic protocols, and we have discovered that this remains true in HISP-based protocols. Message components could be made more explicit by the incorporation of node identities, but of course the role of identities is much weaker in HISPs than in traditional protocols.

An alternative way of eliminating the replay attack would be to substitute k for $hash(k)$ in the final digest, but our digest specification does not tell us that nothing can be inferred about M from $digest(k, M)$ so it may be unwise to digest confidential information.

This exercise has demonstrated that automated protocol analysis is as valuable in the case of HISPs as in other sorts of protocol attacks: it is able to reveal unexpected behaviours and, when successful, to give a high degree of confidence in the protocols involved.

For more information about our CSP model of protocol analysis the reader can look at [29] and [2]. A number of CSP files that illustrate the techniques we discuss can be downloaded from [1] so that the reader can run them and adapt them for other protocols. These include a group of files investigating the aspects of the key agreement/payment protocol that we have discussed in this section.

¹⁰In the mobile banking variant discussed earlier there is no payment message from customer to merchant, rather the payment goes through the banking system.

¹¹The example file supporting this paper splits it into two: hk_C and $\{k, hash(1 : hk_M)\}_{pk}$. In cases like this where a message consists of the concatenation of two or more parts, rather than being the result of a cryptographic operation, it is always legitimate to make this transformation: it does not change the security properties of the protocol. The reason for the transformation is to reduce the sizes of the types that a tool like FDR has to consider: the alphabet is cut from 43,668 to 12,920 in this case.

6.4 The prospects for stochastic analysis

In order to follow the second approach to checking these protocols, based on stochastic analysis, we need a tool that can calculate appropriate probabilistic properties of discrete systems. We need to build the stochastic model of digest functions in terms of the digest collision probability ϵ that we set out earlier into such a tool. We are presently investigating such tools as Prism [5] for this purpose.

We think that stochastic analysis is more likely to be tractable, and beneficial, on protocols where the more conventional symbolic methods above find no attack rather than where they do. There is probably not that much point in doing a detailed stochastic analysis on a protocol where we know there is an attack.

7 Digest functions

In the preceding sections we have assumed the existence of a digest function with given properties. In this section we show that this can be realised relatively simply in such a way that it can be implemented efficiently on all microprocessors. We give below a slightly more exact definition of a digest function, where M , K and b are the bitlength of message, key and output in a digest function, and $R = \{0, 1\}^K$, $X = \{0, 1\}^M$ and $Y = \{0, 1\}^b$.

Definition 1. [21] A (ϵ_d, ϵ_c) -balanced digest function, $digest : R \times X \rightarrow Y$, must satisfy

- for every $m \in X \setminus \{0\}$ and $y \in Y$: $\Pr_{\{k \in R\}}[digest(k, m) = y] \leq \epsilon_d$
- for every $m, m' \in X$ ($m \neq m'$): $\Pr_{\{k \in R\}}[digest(k, m) = digest(k, m')] \leq \epsilon_c$

It is possible create a digest by careful processing of standard cryptographic hash functions, as detailed in [11, 14]. But intuitively this seems wasteful: it should not be necessary to create, expensively, say a 256-bit hash only to discard most of this information immediately. We might expect that more efficient direct algorithms can be found, and indeed this is so. Digests, and therefore the computation of them, are closely related to the idea of a *universal hash function*, as discussed in [8, 34].

In the following we describe a digest algorithm introduced by us in [21] which uses word multiplications to obtain a very fast implementation.

Let us divide message m into b -bit blocks $\langle m_1, \dots, m_{t=M/b} \rangle$. A $(M + b)$ -bit key $k = \langle k_1, \dots, k_{t+1} \rangle$ is selected randomly from $R = \{0, 1\}^{M+b}$. A b -bit $digest(k, m)$ is defined as

$$digest(k, m) = \sum_{i=1}^t [m_i * k_i + (m_i * k_{i+1} \text{ div } 2^b)] \text{ mod } 2^b \quad (1)$$

Here, $*$ refers to a word multiplication of two b -bit blocks which produces a $2b$ -bit output, whereas both $+$ and \sum are additions modulo 2^b .

This scheme is related to the multiplicative method of Dietzfelbinger et al. [10], as illustrated by Figure 1 where all word multiplications involved in Equation 1 can be arranged into the same shape as the overlap of the expanded multiplication between m and k . This algorithm enjoys strong and provable security properties as shown by the following theorem.

Theorem 1. [21] For any $t, b \geq 1$, $digest()$ of Equation 1 satisfies Definition 1 with the distribution probability $\epsilon_d = 2^{-b}$ and the collision probability $\epsilon_c = 2^{1-b}$ on equal length inputs.

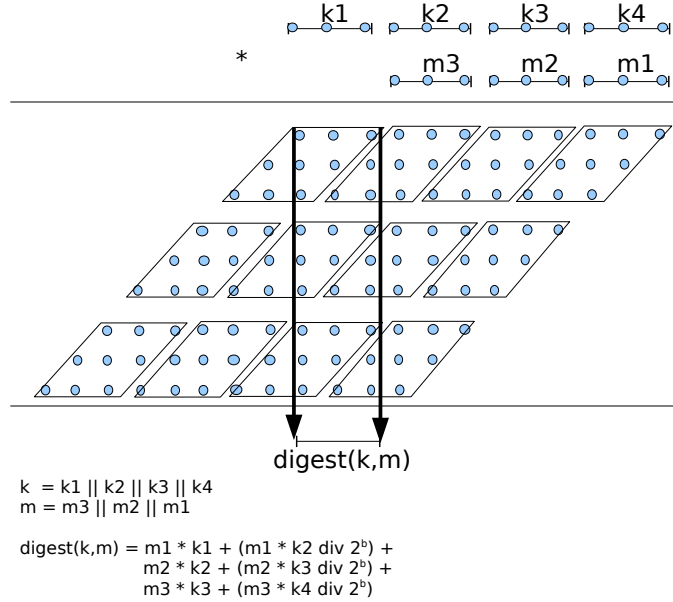


Figure 1: Word multiplication model $digest(k, m)$. Each parallelogram equals the expansion of a word multiplication between a b -bit key block and a b -bit message block.

Although single-word or b -bit digest schemes where $b \in \{8, 16, 32\}$ are suitable in many applications, we might need to reduce the collision probability without increasing the value of b that is dictated by architecture characteristics. It turns out that a single-word digest scheme can be straightforwardly generalised to a multiple-word (or nb -bit) digest construction $digest_{MW}()$, where MW stands for multiple-word. This is graphically illustrated in Figure 2.

We still divide m into b -bit blocks $\langle m_1, \dots, m_{t=M/b} \rangle$. However, a $(M + bn)$ -bit key $k = \langle k_1, \dots, k_{t+bn} \rangle$ will be chosen randomly from $R = \{0, 1\}^{M+bn}$ to compute a nb -bit digest.

For all $i \in \{1, \dots, n\}$, we then define:

$$d_i = digest(k_{i \dots t+bn}, m) = \sum_{j=1}^t [m_j k_{i+j-1} + (m_j k_{i+j} \text{ div } 2^b)] \text{ mod } 2^b$$

And

$$digest_{MW}(k, m) = \langle d_1 \dots d_n \rangle$$

The following theorem shows that $digest_{MW}()$ enjoys the best bound for both collision and distribution probabilities that one could hope for.

Theorem 2. [21] For any $n, t \geq 1$ and $b \geq 1$, $digest_{MW}()$ satisfies the definition of a digest function with the distribution probability $\epsilon_d = 2^{-nb}$ and the collision probability $\epsilon_c = 2^{n-nb}$ on equal length inputs.

For completeness, we give results on the implementations of $digest()$ with respect to different output lengths as well as SHA family of hash functions on a workstation with a 1GHz AMD Athlon(tm) 64 X2 Dual Core Processor in Tables 1 and 2.

Output bitlength	32	64	96	160	256
ϵ_c	2×2^{-32}	$2^2 \times 2^{-64}$	$2^3 \times 2^{-96}$	$2^5 \times 2^{-160}$	$2^8 \times 2^{-256}$
Speed (cycles/byte)	1.57	1.93	2.36	2.79	2.89

Table 1: Performance (cycles/byte) of *digest* constructions, which include the cost of both pseudorandom key generation and digest computation.

1GHz AMD Athlon 64 X2	SHA1: 5.78	SHA256: 12.35	SHA512: 8.54
-----------------------	------------	---------------	--------------

Table 2: Speed of SHA hash functions (cycles/byte).

8 Implementation

In this section we describe some demonstrator implementations of electronic payment using our technology. Films of these in action can be seen (at the time of writing) at www.cs.ox.ac.uk/hcbk.

Our initial implementation (labelled low-power on the web site) was of a protocol for credit card transactions using an e-cheque. The SD is a board with a Teridian 73S1217 processor (with 80515 8-bit core), a card reader, LED display and simple keyboard, and provides a good model of a cheap card-reading device or a future chip on a card. It has just 2K of data memory.

The implementation is written in a mixture of C and assembly code, and uses 1024-bit RSA (with public exponent 65537), 128-bit AES and the SHA-1 hash function. The only one of these that is time critical is RSA, for which our single encryption takes approximately 3 seconds on the above platform. (The decryption, which will almost always involve a more complex exponent, would take much longer on this platform, but fortunately our protocol means that the decryption takes place on the merchant’s computer, not our-low power device.) We experimented with Diffie-Hellman and ECC as alternatives to RSA, but at this level of security we found that RSA was faster on our simple SD thanks to the use of a low exponent. This implementation runs in conjunction with two PCs: one modelling the customer’s PC which is connected to the merchant’s via an `https` session. The customer’s PC is used as an untrusted intermediary between SD and merchant, who displays his check-string on the `https` window.

Essentially the same implementation runs on mobile phones (though without a card reader). These, of course, have much greater computational power and memory than the 8-bit microprocessor, and our main motivation for this work was to experiment with a wide variety of ways of connecting SD and merchant. We have used a very wide variety of methods, including sound and digital telephony, SMS, wifi, Bluetooth and the Internet (via a central server that it is not necessary to trust). Some of these were in combination with the user’s PC as an untrusted intermediary, as above and shown in the film *On-line payment*. All methods worked well with the exception of SMS messaging (a separate message for each protocol message), since these simply take too long to arrive through the telephone system.

The third film is labelled *Peer to peer payment*. In this, two mobile phones are connected via Bluetooth: the protocol will start after the Bluetooth discover-and-connect process. An e-cheque is sent to the payee from the payer. To simplify the demonstration, we do not show a second connection to a bank or a third party. This is implemented on Nokia N95 and Blackberry 9000: a J2ME Midlet is programmed to run on N95, and a JAVA (on RIM)

application is programmed to run on Blackberry 9000. The Bluetooth is v2.0 and the profile is no security.

The following pictures show a few details of the above.

The mobile phone implementation has been extended beyond the pairing required for financial transactions to enable a group of these two be formed. We can presently (shown in a further video on the web site) form mutually authenticated groups of 4 in around 30 seconds, and expect to be able to reduce this to 15s.



Figure 3: Demonstrations: SD with card reader, phone-to-phone, phone-to-server (from left to right).

9 Conclusions

We have seen how, in transactions involving Alice proving her identity to some party Bob whom she can identify by context, it often makes sense for her to get a connection that she knows is with Bob, even if she does not know Bob's name. She can then use that connection to prove her identity securely, and perhaps perform other functions, and has no need to place a credit card, identity card etc in the hands of another party, thereby enabling her to control what information is taken. In other words, before she authenticates herself in one direction, she performs an authentication in the *reverse* direction.

We believe that this technology will have many applications both within the area of financial transactions highlighted here and more widely.

We also believe that HISPs will, with time, allow security to become more organic, based on human trust rather than on some mysterious infrastructure that most users neither understand nor use properly. Who could put their hand on their heart and say they had never over-ridden a security certificate failure on their computer without thoroughly checking the certificate? We are certainly not arguing that they will take over from PKIs and conventional forms of authentication, but rather that they will become common in areas where one or more humans have to form a pair or group of devices. Indeed we have demonstrated here that HISPs can allow conventional security to be bootstrapped more efficiently and with less leakage of private information.

Since HISPs fundamentally require the participation of humans, it is vital that their implementations are designed to optimise people's experience in using them, are designed to avoid users complacently ignoring the requirements on them, and make it easy to compare a reasonably secure digest or check-string length quickly. Research is ongoing into these human factors issues [18, 26, 32].

As we have seen, considerable progress is being made on the implementation, verification and novel cryptographic needs of these protocols. But the fact that so much work is necessary,

and that the whole idea of trust needs to be rethought, demonstrates just how exciting this area is.

References

- [1] CSP files that illustrate the techniques we discuss in this paper can be downloaded from:
<http://www.cs.ox.ac.uk/publications/publication5348-abstract.html>
- [2] CSP files that illustrate the techniques we discuss in [29] can be downloaded from:
<http://www.cs.ox.ac.uk/publications/publication5265-abstract.html>
- [3] See: <http://www.cs.ox.ac.uk/gavin.lowe/Security/Casper/>
- [4] See: http://en.wikipedia.org/wiki/Oyster_card
- [5] See: <http://www.prismmodelchecker.org/>
- [6] M. Abadi and R. Needham. *Prudent Engineering Practice for Cryptographic Protocols*. IEEE Transactions on Software Engineering, Volume 22 Issue 1, January 1996.
- [7] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, P. Rogaway. *UMAC: Fast and Secure Message Authentication*. CRYPTO, LNCS vol. 1666, pp. 216-233, 1999
- [8] J.L. Carter and M.N. Wegman. *Universal Classes of Hash Functions*. Journal of Computer and System Sciences, 18 (1979), 143-154.
- [9] S.J. Creese, M.H. Goldsmith, A.W. Roscoe and I. Zakiuddin. *The attacker in ubiquitous computing environments: Formalising the threat model*. Workshop on Formal Aspects in Security and Trust, 2003.
- [10] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. *A reliable randomized algorithm for the closest-pair problem*. Journal Algorithms, 25:19-51, 1997.
- [11] C. Gehrman, C. Mitchell and K. Nyberg. *Manual Authentication for Wireless Devices*. RSA Cryptobytes, vol. 7, no. 1, pp. 29-37, 2004.
- [12] S. Halevi and H. Krawczyk. *MMH: Software Message Authentication in the Gbit/second Rates*. FSE, LNCS vol. 1267, pp. 172-189, 1997.
- [13] S. Laur and K. Nyberg. *Efficient Mutual Data Authentication Using Manually Authenticated Strings*. Volume 4301 on LNCS, 90-107, 2006.
- [14] A.Y. Lindell *Comparison-Based Key Exchange and the Security of the Numeric Comparison Mode in Bluetooth v2.1*. In the Proceedings of RSA Conference 2009, 66-83.
- [15] G. Lowe. *Analysing Protocol Subject to Guessing Attacks*. Journal of Computer Security 12(1): 83-98 (2004).
- [16] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone. *Handbook of Applied Cryptography*. ISBN: 0-8493-8523-7.
- [17] M. Matsumoto and T. Nishimura. Fast Mersenne Twister Homepage where the publicly available source code for this pseudorandom number generator can be found:
<http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/SFMT/>

- [18] J.M. McCune, A. Perrig and M.K. Reiter. *Seeing is Believing: Using Camera Phones for Human-Verifiable Authentication*. IEEE Symposium on Security and Privacy, 2005, 110-124.
- [19] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone. *Handbook of Applied Cryptography*. ISBN: 0-8493-8523-7.
- [20] L.H. Nguyen (editor), second edition. ISO/IEC 9798-6 (2010): *Information Technology – Security Techniques – Entity authentication – Part 6: Mechanisms using manual data transfer*.
- [21] L.H. Nguyen and A.W. Roscoe *On the construction of digest functions for manual authentication protocols*. Submitted for publication.
- [22] L.H. Nguyen and A.W. Roscoe. *Efficient group authentication protocol based on human interaction*. In Proceedings of FCS-ARSPA 2006, 9-31.
- [23] L.H. Nguyen and A.W. Roscoe. *Authenticating ad hoc networks by comparison of short digests*. Information and Computation 206 (2008), 250-271.
- [24] L.H. Nguyen and A.W. Roscoe. *Authentication protocols based on low-bandwidth unspoofable channels: a comparative survey*. Journal of Computer Security (64 pages), to appear in 2011.
- [25] L.H. Nguyen and A.W. Roscoe. *Separating two roles of hashing in one-way message authentication*. Proceedings of FCS-ARSPA-WITS 2008, 195-210.
- [26] R. Kainda, I. Flechais and A.W. Roscoe. *Usability and Security of Out-Of-Band Channels in Secure Device Pairing Protocols*. In the Proceedings of SOUPS 2009.
- [27] A.W. Roscoe. *The theory and practice of concurrency*. Prentice Hall. 1998. ISBN-10: 0136744095. ISBN-13: 978-0136744092.
- [28] A.W. Roscoe. *Human-centred computer security*. (2005) See: web.cs.ox.ac.uk/oucl/work/bill.roscoe/publications/113.pdf
- [29] A.W. Roscoe, T. Smyth and L.H. Nguyen. *Model checking cryptographic protocols subject to combinatorial attack*. Please see <http://www.cs.ox.ac.uk/files/4157/guess.pdf>
- [30] P. Ryan (Author), S. Schneider, M. Goldsmith, G. Lowe, A.W. Roscoe. *Modelling and Analysis of Security Protocols*. Addison-Wesley Professional. 2000. ISBN-10: 0201674718. ISBN-13: 978-0201674712.
- [31] M. Saito and M. Matsumoto. *SIMD-oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator*. Monte Carlo and Quasi-Monte Carlo Methods 2006, Springer, 2008, pp. 607- 622.
- [32] J. Suomalainen, J. Valkonen and N. Asokan. *Security Associations in Personal Networks: A Comparative Analysis*. Vol. 4572 on LNCS, 43-57, 2007.
- [33] S. Vaudenay. *Secure Communications over Insecure Channels Based on Short Authenticated Strings*. Advances in Cryptology - Crypto 2005, LNCS vol. 3621, 309-326.

- [34] M.N. Wegman and J.L. Carter. *New Hash Functions and Their Use in Authentication and Set Equality*. Journal of Computer and System Sciences, 22 (1981), 265-279.