Key agreement via protocols

A W Roscoe and Wang Lei

University College Oxford Blockchain Research Laboratory

March 7, 2019

Abstract

Inspired by the ideas of *no cloning* and *measurable degrading* relied on by quantum key agreement protocols, we devise novel key agreement protocols for the classical world that might be run by mass produced identical devices, or parties using them. We thus use protocols a little outside their normal range and seemingly achieve the impossible by relying on assumptions that may or may not be reasonable.

1 Introduction

Cathy Meadows is a truly seminal figure in the field of cryptographic protocols, having contributed both directly through her own work and though the many people that she and her work [8, 9, 10] have inspired, to the development and understanding of this subject. And of course her work on the automation of the analysis of protocols has seen the creation of protocols develop from something that was close to a black art and certainly immensely risky to one where there are a range of sophisticated and powerful tools for creating and analysing them. She was already well established in this field when, together with Gavin Lowe and Michael Goldsmith, Roscoe joined in in 1994/5 [6, 11, 14, 7]. Her friendship and advice have been a constant inspiration ever since.

Cryptography and cryptographic protocols are both powerful tools for attaining security goals. Initially, at least, these two communities stayed rather separate, with the protocol community relying on perhaps simplistic assumptions such as "perfect cryptography", gradually expanded to encompass finitely presented algebraic weaknesses in cryptosystems such as Vernam and RSA. They seek wide ranging secure goals including the sharing and authentication of keys, privately when for symmetric constructs, achieving transactions such as the transfer of value, voting or consensus, or achieving a linked and authenticated data stream. Cryptographers have sought lower level proofs about the cryptographic constructs themselves. Historically there have not been sufficient links between the two groups, though there have been better integration recently through [1, 5] etc.

In this paper we show a case in which it is possible, under assumptions, to replace cryptography that we do not want to use by protocols that only use totally standard symmetric ciphers and hash functions. In a strong sense we are moving a problem from the world of cryptography to the world of cryptographic protocols.

Our goal is to find replacements for asymmetric constructs such as public key cryptosystems, associated signature and Diffie Hellman variants. The reasons for wanting to do this are well known:

1 These forms of calculation are often too resource hungry for lightweight applications such as IoT nodes.

2 Quantum computers are known [15] to undermine the best known such methods once built, and our knowledge of what they can solve is poor. There is much to be said for not relying on the post quantum security of anything beyond hashing and symmetric crypto. The most convincing argument for this is that if either of these were to go it seems as there would be no basis left at all for creating communications security. It is very possible that a post-quantum cryptosystem that went beyond this would be vulnerable to novel quantum algorithms. And of course if a new model of post-quantum cryptography based on lattices etc turned out to be less efficient in computation and space than Diffie-Hellman, RSA etc, then the need for

Both of these provide us with good reason for following the aqproach we do.

In other words we want to build asymmetric encryptions, secret key exchange and efficient signature mechanisms using protocols created from generic and sufficiently long cryptographic hash functions, and sufficiently secure symmetric encryptions.

Notwithstanding the possibilities of Merkel puzzles and similar, which (given that they are based on searching) are in fact probably vulnerable to quantum computers through Grover's algorithm, offer little security, it is seemingly impossible to achieve the above goals within standard models of how protocols and their implementations are constructed.

It seems that we need some clever idea or *deus ex machina* to help us. The one we choose is that we rely on hardened physical security of the devices the protocols are implemented on. Specifically we assume we can create devices meeting the following specifications.

a The device provides a simple well defined cryptographic and perhaps protocol service identically for all.

b Perhaps contain a key, identical in all devices, that is available to the above service to use.

c The key cannot me extracted by an attacker

d No unadvertised functionality of the device is obtainable by anyone.

Conditions c and d amount to a pretty strong assumption of the device not being vulnerable to reverse engineering. In this paper we do not comment on how easy or difficult it may be to attain these goals, other than to say that one would naturally expect that the more expensive and customised the manufacturing technology is, the better the resistance will be. Even if the individual devices are moderately costly, the benefit of having them all identical and not requiring the post-manufacture or individual initialisation and management of keys will be considerable.

Resistance to reverse engineering is a popular research topic in areas like digital rights mechanism, and has various technologies that implement degrees of it in general such as Hardware Security Modules which tend to be elaborate and expensive, and SGX enclaves on Intel processors.

The possibly lower security threshold and certainly low energy threshold for IoT makes this the most likely initial application, in the authors' opinion.

For the purposes of the present paper the reader is asked to put this sort of question on one side and treat the core problem as an intellectual exercise. In essence, in a long mathematical tradition, we are reducing one difficult problem to another.

Our assumptions imply that no *a priori* key management mechanisms are required. We can rely on them working out their own keys *in situ.* This is a huge and attractive simplification.

2 Background

Asymmetric cryptography provides the core that supports most of modern security by establishing secret keys and implementing signature. Unfortunately the two problems highlighted in the introduction mean we have to look elsewhere than the traditional methods which become expensive as the power of the conventional attacker grows and quantum computers that can run Shor's and Grover's algorithms get closer.

The second of these has generated a great deal of research on models of signature and asymmetric cryptography that are not thought to be vulnerable to quantum computers. It is generally believed that standard means of cryptographic hashing are or can be made invulnerable, and strong enough symmetric cryptography such as AES similarly, provided a small multiple (typically 1.5 or 2 depending on the detail) is applied to the number of bits involved. However these do not obviously solve the problem since they do not obviously provide an alternative. Much of the work is devoted to lattice-based cryptography, seen by some as the most promising asymmetric prospect. This paper concentrates on uses of conventional primitives in non-standard ways.

The methods we describe are extremely efficient in the amount of cryptographic calculation required and therefore offer prospects of security to applications such as IoT where asymmetric cryptography is barred on cost grounds rather than only because of the worry of future quantum computers.

3 Key agreement

The concepts below were developed from one of the key ideas underpinning quantum key exchange [2], namely that it is impossible to clone a particle passed from one party to another without disturbing the system, thereby making it and similar activity by an attacker discoverable. This led to the question of how you might create, and exploit, non-clonable transmissions in the classical world.

We imagine that the world is populated by *widgets* that communicate at two levels: with their local users over secure channels (i.e. ones that we do not have to make secure), and through the ether (such as internet or radio waves) which we suppose to be run on Dolev-Yao lines where messages can be copied, blocked or faked at will. To counter this we imagine that the widgets are built to resist reverse engineering and out-of-spec use and so will not reveal their internal secrets.

All these widgets are identical. If A and B hold one each and want to exchange a key secretly with each other via their widgets, how can they do so and be sure they have not been talking to Eve instead, and know that Eve cannot have acted like a man-in-the-middle or have used one of more of her own widgets to obtain their key?

We are trying to design ones that allow a pair of nodes that own them the ability to run a protocol that gives them the equivalent of running Diffie-Hellman in the conventional pre-quantum world.

We must make widgets able to talk in a way which is secret (to all but other suitably prepared widgets) and such that they guarantee no cloning of messages. This solves some of the problems above.

To do this we will assume they use a secret ms (master secret) that is known to all widgets and no-one else. We could use this to encrypt things X they want to send to each other as $\{X\}_{ms}$. However this would mean that there was a lot of traffic encrypted under ms creating the danger of it being revealed by cryptanalysis. We will therefore use it in more subtle ways.

Because all widgets are identical, Alice cannot tell whether she is talking to Bob's or Eve's. There is no need for us to worry about this, because this is a familiar situation. If Alice was using Diffie-Hellman then that would not of itself tell her who she was running it with. With Diffie-Hellman we need to use authentication so Alice and Bob can confirm that they were talking with each other, and the same will be true here. What we need to confirm is that if Alice and Bob do share a key through use of the widgets, then no-one else shares it. What we want, in effect, is that our widget provides, in a post-quantum world, the same guarantees that doing Diffie-Hellman computation provides in a classical one.

In the world of Diffie-Hellman, this can be achieved by having Alice and Bob agree over an authentic but not private channel on the hash of the key they have developed. A belt-and-braces approach would for be for them to include not only the key but also their g^x and g^y and p and g in this hash. We can clearly adopt an analogous approach here, or use some (perhaps hash-based) cryptographic signature to authenticate common ownership.

Imagine first that Alice simply wishes to create a new key `X` and send it securely to partner Bob via such widgets `A` and `B` (which do not have actual identities, but are the ones they own).

```
Protocol L1

0. Alice -> A: X
1. A -> B    : go
2. B -> A    : {NB}_ms          NB is a fresh random nonce
3. A -> B    : {X,NB}_ms
4. B -> Bob  : X
```

This would be followed by a phase in which Alice and Bob agree over another authentic channel that they have both got, say `hash(X)`. (This confirms but does not reveal `X`.)

Here messages 0 and 4 are the message being communicated from A's user Alice to A, and B to B's user Bob. *go* is just a signal to B to start. It then sends N_B, a random fresh nonce, which is included by A with X in Message 3, with B refusing to accept this message unless its own nonce is included. (The names A and B are included here for clarity to us: they do not really exist and the widgets do not know them, at least in any secure sense). All communications between a user and his/her widget are assumed to be secure.

Depending on the implementation `A` and `B` might communicate directly via the Dolev-Yao medium or might have Alice and Bob do so for on their behalves.

We assume that as soon as the value X or N_B has performed its role in the protocol for A or B, the widget forgets it, and will not perform any other series of messages than those required to perform the sender or receiver role in the above. In particular Alice's will never re-use N_B and Bob's will accept at most one Message 3 for each N_B it generates.

For any such protocol we require that the sender A only sends X on once it has received an entropy from the receiver B, and that this send of X is both confidential and only successful when this entropy is bound up with X in some chosen way. The above is one way of achieving this; we will see more later. The entropy must be fresh and unguessable, and the data must be forgotten by the nodes as soon as possible, and not re-used.

The above protocol is not ideal because encryption under ms is over-used. One way of fixing this is

```
Protocol L2

0. Alice -> A: X
1. A -> B    : NA              NA is a fresh random nonce
both widgets compute ks = hash(NA,ms)
2. B -> A    : {NB}_{ks}       NB is a fresh random nonce
3. A -> B    : {X,NB}_{ks}
4. B -> Bob  : X
```

The guarantee provided by such a protocol is that any X a node sends reaches the user of at most one other widget. Eve can divert the messages from B to E, which will lead to her receiving X. But she cannot persuade both B and E to reveal X, because to do this she would need to get X paired with two different nonces N_B and N_E. And A pairs X with one and forgets it. Note that this contributes towards the no-cloning of X.

Thus using these, Alice knows that any X she sends reaches at most one party, who might be Bob, and Bob knows that any he received came from someone, who might be Alice. This is very like the logical properties of Diffie-Hellman: anyone who runs the interchange through their widget knows they share a secret `X` with someone, but does not know who.

Crucial to understanding these protocols is that the widget B generates a unique ticket for each session, namely `NB`. It will never accept more than one message with any given ticket, and no widget will accept a message with any ticket other than the one it has itself generated.

If we assume that Alice and Bob have an authentic but not secret channel between them, namely one that can be overheard but where they know conclusively that they are talking to each other, they can compare the hashes of an X sent by Alice and one received by Bob to check they are the same. Does this prove that no-one else other than the two of them knows X (on the assumption that it is an entropy created by Alice that is unguessable)?

The answer to this is "no" because we have not eliminated the possibility of a man in the middle (MITM). Eve might own two widgets E and F. She gets E to obtain X from A and then has F send the same X to B. She then knows X even though Alice and Bob agree on it. How can we prevent this? In essence the protocols and our assumptions about widgets ensure that X cannot be copied inside the world of widgets: the above protocols are *linear* in the same sense as linear logic, but they do not prevent copying X outside this realm and passing it on.

It follows that protocols L1 and L2 do not meet our requirments: they do not guarantee secure key exchange.

Again taking a clue from the quantum key exchange world, one solution to this is to have A and/or B degrade `X` in such a way that the test of agreement assumes a single degrading, but does not permit two.

Two interesting forms of degrading are:

- Hashing: if the protocol is modified so that widget B outputs `hash(X)` rather than `X`, then the MITM attack above hashes `X` twice. If Alice and Bob compare `hash(hash(X))` with `hash(Y)`, where `Y` is B's output, they can easily detect the MITM. The approach to sharing a key could then be that Alice sends a pre-key random value `X` to A, with the hope that `k = hash(X)` will be delivered to Bob by B. The actual key is then `k`. (Alternatively the value `X` can be hashed just by A or by both widgets. If both then the check would have to be different and the natural key would be `hash(hash(X))`.)

- Delay: node B delays the last message by interval T. Bob only accepts

X if received $< 2T$ from Alice's send. This time can be sent authentically from Alice to Bob over the same channel used for comparing the keys at either end as part of the key checking. Note that this mechanism allows any message to passed from Alice to Bob, rather than the scrambled $hash(X)$ of the first.

Both of these provide the opportunity for Alice and Bob to exclude the man in the middle on the assumption that they sent/received their messages via a widget, and that the only thing that can be done with a widget is to use it according to its functional specification.

We are thus relying that, even though Eve may own a number of widgets identical to Alice's and Bob's, she cannot use these to decrypt the messages that go between Alice's, Bob's or anyone else's widgets other than by performing a complete protocol, and she cannot extract anything other than the intended outputs from them.

With the use of hashing as degrading the above protocols become

```
Protocol KA1

0. Alice -> A: X
1. A -> B    : go
2. B -> A    : {NB}_ms        NB is a fresh random nonce
3. A -> B    : {X,NB}_ms
4. B -> Bob  : hash(X)        hash computed by B
                              Bob has no access to X.
```

Of course the above is sub-optimal because it over uses encryption under `ms`.

```
Protocol KA2

0. Alice -> A: X
1. A -> B    : NA             NA is a fresh random nonce
both widgets compute ks = hash(NA,ms)
2. B -> A    : {NB}_{ks}      NB is a fresh random nonce
3. A -> B    : {X,NB}_{ks}
4. B -> Bob  : hash(X)        hash computed by B
                              Bob has no access to X.
```

The `KA` in these names stands for key agreement. In each case we assume that A is obliged to invent a new fresh entropy X for each key she wants to share.

The delay variant of these protocols is intriguing because it allows secret transmission from Alice to Bob without them even having to know keys. The only problem is that Alice has to check that it really is Bob she has shared the secret with since she cannot be sure in advance.

4 Reducing the role of the widget

What is it essential for a widget to do? The ones implied in the previous section need to hold on to a long-term secret, and perform specified calculations utilising that secret. These things appear to be necessary. However they also need to hold a certain amount of state, generate random numbers, and hold onto temporary variables such as the nonce `NB`. These things do not seem so obviously necessary, and when we are trying to build something that cannot be reverse engineered it is as well to keep it as simple as possible.

Let us first examine the issue of random numbers: `NB` is generated by B in both protocols and in the second one A also generates `NA`. `NA` is used to generate a fresh key for sending `NB`. Strictly speaking, it does not seem to be necessary to encrypt `NB` at all in these protocols: its role in the protocols is, in essence, to prevent any other recipient E accepting the message that A sends to B in this protocol. Another E would not have generated this `NB`.

The following is based stripping the protocols above to the bone, and leaving all the creation of random numbers and remembering of them to ALice and Bob as opposed to the widgets. The biggest trick here is making sure that `NB` can still only be used by Bob to obtain the transmission, and never allows any other agent to get at it.

```
Protocol KA3

1. Alice -> Bob: go
3. Bob -> Alice  : hash(NB)
3. Alice -> A:(X,hash(NB))
        ks = hash(hash(NB),ms)
4. A -> B    : {X,hash(NB)}_{ks}
4a: Bob -> B : NB   enabling B to compute ks and decrypt Message 3
        X (in B) := this decryption
5. B -> Bob  : hash(X)        hash computed by B
```

Here is is Bob's duty to ensure the freshness and secrecy of `NB`: if he re-uses a value then the no-copying property does not hold.

This results in widgets that do not need state, all they have to do is

- At A: Given `(X,hash(NB))` compute `{X,hash(NB)}_ks` in an atomic fashion.
- At B: Given encryption `Y` and `NB`, compute `decrypt(hash(hash(NB,ms)),Y)` in an atomic fashion.
- In both cases with no intermeditate calculations or data availabile.

We are thus getting Alice and Bob to remember the state, not the widgets. The important difference here is that we are making B decrypt the packet under the still-secret NB, which has never been seen by anyone other than Bob before. So here `NB` is acting as a secret key that Bob retains for himself to unlock the message he has triggered with `hash(NB)`. Because no other widget ever knows `NB`, no-one else can get at the value `hash(X)`.

The rationale here for degrading `X` to `hash(X)` are exactly the same as they were with `KA1` and `KA2`.

The basic rationale for degrading `NB` to `hash(NB)` in most of this protocol is essentially the same as the rationale for degrading `X` to `hash(X)`: it prevents an attack based on copying. This seems to what is required to move from a situation where A or B is trustily generating random values to one where they accept values from their users who might not be so trustworthy in general. One might infer that if we did not let A input a pre-key from Alice ubt instead create it internally, then degrading would not actually be necessary at all.

This is witnessed by the following protocol, which requires no degrading.

```
Protocol KA4

2. B -> A    : NB             NB is a fresh random nonce
     ks := hash(NB,ms)
3. A -> B    : {X,NA}_{ks}    NA freshly created by A
4. B -> Bob  : NA xor NB      (or NA)
4a A -> Alice  : NA xor NB      (or NA)
```

This protocol achieves a shared secret key in a simpler way, in some sense, again using `NB` as a ticket. This seems to work for our purposes with in some sense simpler widget functionality, but where state is needed. We cannot here trust Bob to input `NB` or Alice to input `NA`, so random number generation is needed here, unlike in `KA3`.

We regard `KA3` as the most promising key agreement protocol, because the widgets need no state, and do not need to generate random numbers.

What they do need to do is to keep the long term secret *ms* securely, and implement the compound functions implementing the encryption and decryption phases of the calculation, and not provide any of the component functions making these up.

5 Verification

The protocols described in this paper look very much like the ones that we have been putting through protocol verifiers for years. The main novelties are

- The role of widgets as trusted computing cores for whom we do not trust the owners.
- The lack of reliable identities at a level at which the protocols and widgets are meant to work.

In a CSP/Casper model the natural way to analyse the stateless widgets version of the protocol is to build all of the capabilities of a widget usable for (say) protocol `KA3` into the intruder model. Thus if the intruder posesses `{X,Y}` then we can deduce `{X,Y}_{hash(Y,ms)}` and if it possesses `{X,hash(NB)}_{hash(hash(NB),ms)}` and `NB` then it can deduce `hash(X)`

All this is, of course, in addition to the usual rules of an intruder. Thus stateless widgets seem to fit in well, simply treating the widgets as what they in effect are, a type of oracle.

Widgets that have internal state really need to be treated as trustworthy participants in the protocol, and we need to allow for an unbounded number of them to be present, reason that somme small number such as 2 is sufficient, or prove limited results on how many our models allow for. This situation is, or course, very similar to the problem of reasoning about many parallel sessions of a cryptoprotocol as was done in [12, 3], for example. We leave the resolution of that case to future research.

The question of whether widgets do or do not have internal state also affects how one programs the actions of the likes of Alice and Bob in the network. Just as in their availability to the intruder discussed above, a stateless widget used by a trustworthy agent does not need a separate process to implement it, whereas a stateful one probably does. We would possibly avoid the last issues if Alice and Bob were assumed to own only one widget each.

6 Conclusions and further thoughts

This is at least an interesting thought experiment into how to achieve key exchange in unusual ways. How practical it is will depend on the possibility of resisting reverse engineering.

We have similarly provided approaches to asymmetric cryptography: see [13], where we will go into more detail about reverse engineering. There are existing discussions of this, including [4].

Clearly the ideas developed here are closely related to those of trustworthy computing and trusted execution environments.

We observed that non-stateful widgets will fit smoothly into the establoshed methods of verifying cryptoprotocols, but that stateful ones might pose more of a challenge.

References

[1] Armando, Alessandro, *et al.* "The AVISPA tool for the automated validation of internet security protocols and applications." International conference on computer aided verification. Springer, Berlin, Heidelberg, 2005.

[2] Cerf, Nicolas J., Marc Levy, and Gilles Van Assche. "Quantum distribution of Gaussian keys using squeezed states." Physical Review A 63.5 (2001).

[3] Chevalier, Yannick, and Laurent Vigneron. "Automated unbounded verification of security protocols." International conference on computer aided verification. Springer, Berlin, Heidelberg, 2002.

[4] Brice Colombier, Lilian Bossuet. A Survey of hardware protection of design data for integrated circuits and intellectual properties. IET Computers & Digital Techniques, Institution of Engineering and Technology, 2014, 8 (6), pp.274287.

[5] Cremers, Cas JF. "The scyther tool: Verification, falsification, and analysis of security protocols." International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg, 2008.

[6] Lowe, Gavin. "Breaking and fixing the Needham-Schroeder public-key protocol using FDR." International Workshop on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg, 1996.

[7] Lowe, Gavin. ”Casper: A compiler for the analysis of security protocols.” Journal of computer security 6.1-2 (1998): 53-84.

[8] Meadows, Catherine. ”The NRL protocol analyzer: An overview.” The Journal of Logic Programming 26.2 (1996): 113-131.

[9] Meadows, Catherine. ”Formal verification of cryptographic protocols: A survey.” International Conference on the Theory and Application of Cryptology. Springer, Berlin, Heidelberg, 1994.

[10] Meadows, Catherine. ”Applying formal methods to the analysis of a key management protocol.” Journal of Computer Security 1.1 (1992): 5-35.

[11] Roscoe, A.W. ”Modelling and verifying key-exchange protocols using CSP and FDR.” Computer Security Foundations Workshop, 1995. Proceedings., Eighth IEEE. IEEE, 1995.

[12] Roscoe, A. W., and Philippa J. Broadfoot. ”Proving security protocols with model checkers by data independence techniques.” Journal of Computer Security 7.2-3 (1999): 147-190.

[13] Roscoe, A.W., Wang Lei, Chen Bangdao, New approaches to key agreement and public key cryptography. *In preparation*

[14] Ryan, P., Schneider, S. A., Goldsmith, M., Lowe, G., & Roscoe, A.W.. (2001). The modelling and analysis of security protocols: the csp approach. Addison-Wesley Professional.

[15] Shor, Peter W. ”Scheme for reducing decoherence in quantum computer memory.” Physical review A 52.4 (1995).