

## COMSM2004 : Proofs

B. Warinschi and N.P. Smart

Department of Computer Science,  
University Of Bristol,  
Merchant Venturers Building,  
Woodland Road,  
Bristol, BS8 1UB  
United Kingdom.

January 30, 2009

## Outline

Overview

e-Cash

Commitment Schemes

Zero Knowledge Proofs

Proofs of Security and Random Oracles

Advanced ZK-Protocols

## Signature Scheme Scheme

We would like techniques to **prove** that signature schemes are secure.

To get there we need to take a little detour and introduce

- ▶ **commitment schemes**,
- ▶ **zero-knowledge proofs** and
- ▶ **the random oracle model**.

These also come up when we look at more advanced protocols such as how electronic cash (**e-cash**) works.

Therefore, to motivate these concepts we describe a simple e-cash system.

B. Warinschi and N.P. Smart  
COMSM2004 - Proofs

Slide 1



B. Warinschi and N.P. Smart  
COMSM2004 - Proofs

Slide 2



B. Warinschi and N.P. Smart  
COMSM2004 - Proofs

Slide 3



## Money

Locations for money:

- ▶ in your bank account,
- ▶ under your mattress,
- ▶ in your pocket,
- ▶ in cheques,
- ▶ in money orders,....

There are various definitions of what money is.

- ▶ One of the main problems of certain areas of economics is to define money.

## Digital Money

**Digital money** is the term that we use for the money in your bank account.

Your bank knows a great deal about it:

- ▶ how much you have;
- ▶ where it comes from;
- ▶ where it's going next.

Digital money can help to

- ▶ reduces fraud and it
- ▶ gives a complete audit trail.

## Cash

Like some other forms of money, digital money is not **anonymous**; however, cash gives complete anonymity.

- ▶ Some unscrupulous people like to be paid in cash so that the taxman cannot see the transaction!

Benefits of cash:

- ▶ anonymous;
- ▶ unique and difficult to copy;
- ▶ easily transferable (indefinitely);
- ▶ divisible (i.e. you can get change).

Negative aspects of cash:

- ▶ it's bulky;
- ▶ it's hard to transmit;
- ▶ it's easy to steal.

B. Warinschi and N.P. Smart  
COMSM2004 - Proofs

Slide 4



B. Warinschi and N.P. Smart  
COMSM2004 - Proofs

Slide 5



B. Warinschi and N.P. Smart  
COMSM2004 - Proofs

Slide 6



## e-Cash

It would be nice to have an electronic form of cash with

- ▶ all the benefits of cash itself.

There have been a number of digital cash systems proposed over the last twenty years.

- ▶ Many are based on constructs such as digital signatures, hash function and zero-knowledge proofs.

There are a number of design issues:

- ▶ anonymity versus auditability;
- ▶ single use tokens (to detect copying);
- ▶ payment across the Internet need to be made securely;
- ▶ scalability demands off line processing;

In the next few slides we discuss a philosophical protocol which forms the basis of a number of real protocols.

## Anonymous Money Orders - 1

- 1) : Alice prepares an anonymous money order (MO) for 100 pounds and puts it in an envelope with a piece of carbon paper.
- 2) : The bank signs the envelope, and hence the MO, and deducts 100 pounds from Alice's account.
- 3) : Alice opens the envelope and gives the signed MO to Bob.
- 4) : Bob checks the signature of the bank and takes the money order to the bank.
- 5) : The bank checks the signature and gives Bob 100 pounds.

**Note:** The bank, in step 5, cannot trace the MO back to Alice.

- ▶ This is because it never saw what it was signing.

**Problem:** How does the bank know it is signing a MO for 100 pounds and not 200 pounds?

## Anonymous Money Orders - 2

- 1) : Alice prepares 100 anonymous MOs for 100 pounds and puts them in envelopes, each with a piece of carbon paper.
- 2) : The bank opens 99 of the envelopes and checks they are all for 100 pounds.
- 3) : The bank signs the remaining envelope (and hence the money order) and deducts 100 pounds from Alice's account.
- 4) : Alice opens the envelope and gives the signed MO to Bob.
- 5) : Bob checks the signature of the bank and takes the money order to the bank.
- 6) : The bank checks the signature and gives Bob 100 pounds.

**Problem:** Alice or Bob could create duplicate MOs by copying the MO produced above.

B. Warinschi and N.P. Smart  
COMSM2004 - Proofs

Slide 7



B. Warinschi and N.P. Smart  
COMSM2004 - Proofs

Slide 8



B. Warinschi and N.P. Smart  
COMSM2004 - Proofs

Slide 9



- 1) Alice prepares 100 anonymous MOs for 100 pounds, each with a separate serial number and puts them in envelopes containing pieces of carbon paper.
- 2) The bank opens 99 of the envelopes and checks they are all for 100 pounds and that they all have separate serial numbers.
- 3) The bank signs the envelope, and hence the MO, and deducts 100 pounds from Alice's account.
- 4) Alice opens the envelope and gives the signed MO to Bob.
- 5) Bob checks the signature of the bank and takes the money order to the bank.
- 6) The bank checks the signature and gives Bob 100 pounds after checking that the bank has not seen that serial number before.

**Problem:** If duplicate serial numbers are obtained by the bank, who does the bank accuse of cheating, Alice or Bob?

- 1) Alice prepares 100 anonymous MOs for 100 pounds, each with a separate serial number and puts them in envelopes containing pieces of carbon paper.
- 2) The bank opens 99 of the envelopes and checks them as before.
- 3) The bank signs the remaining envelope and deducts 100 pounds from Alice's account.
- 4) Alice opens the envelope and gives the signed MO to Bob.
- 5) Bob asks Alice to write a random identity string on the MO.
- 6) Bob checks the bank's signature and presents the money order to the bank.
- 7) The bank opens the envelope, checks the signature and gives Bob 100 pounds after checking that the bank has not seen that serial number before.

**Note:** If the serial number is a duplicate and the identity strings are the same then Bob cheated, otherwise Alice must have cheated.

The previous protocol still has a number of problems:

It assumes:

- ▶ Bob cannot change the identity string once Alice has written it;
- ▶ Alice cannot frame Bob by using the same identity string twice without Bob noticing;
- ▶ if Bob cheats then the bank knows his identity; and
- ▶ if Alice cheats the bank cannot find out who Alice is, only that someone cheated.

We need a way for the identity string to reveal Alice's identity if, and only if, she cheats.

## Identity Splitting

We need a method in which

- ▶ Alice's identity is kept secret unless
- ▶ she cheats when it is revealed.

This is done using a variant of a **zero-knowledge protocol**.

Alice creates an identity string containing her details: name, address, etc.

- ▶ This is split into two (or many) pieces.

Each piece is **committed to** on the money order before she sends it to the bank for signing.

- ▶ Alice cannot change her details (hence commitment).
- ▶ She can reveal (**de-commit**) what she committed to and it can be verified that she has not cheated.

## Identity Revealing

If Alice spends the same money order twice the bank will detect this and want to recover Alice's identity.

With two such money order's there is a high probability that, for some  $i$ , the bank obtains both  $L_i$  and  $R_i$  - since for the two money orders with the same serial number Bob will have given two distinct bit vectors.

For example

- ▶  $L_i$  from the first use of the money order and
- ▶  $R_i$  from the second use of the money order.

In this case the bank recovers Alice's identity:  $ID = L_i \oplus R_i$ .

## A Commitment Scheme

On the previous slide we required a method for Alice to commit to a string.

Here we use a commitment scheme based on a hash function; we will see more details of commitment schemes in general later in the course.

Suppose we have a bit string  $b$  to which Alice wishes to commit.

- ▶ Alice generates a random strings  $P$ .
- ▶ Alice computes  $h = H(P||b)$ , where  $H$  is a cryptographic hash function.
- ▶ Alice publishes  $h$ .

For Alice to **open** the commitment or **de-commit** she proceeds as follows.

- ▶ Alice supplies  $P$  and  $b$  to Bob.
- ▶ Bob can then check that  $h$  in the commitment equals  $H(P||b)$ .

## Blind Signatures

The only thing we have not discussed is how one implements, digitally, an **envelope with carbon paper** for the bank to sign.

This is accomplished using a **blind signature scheme**.

- ▶ This allows the bank to sign something without knowing what that something is.

To do this we exploit the **homomorphic** property of RSA signatures.

- ▶ Notice that homomorphic property is often considered to be a **bad thing**.

## Identity Encoding

We now return to how Alice encodes her identity on the money order whilst still remaining anonymous (unless she cheats). Alice splits her identity,  $ID$ ,  $n$  times

$$ID = L_i \oplus R_i.$$

She commits to the  $L_i$  and  $R_i$ , without revealing  $L_i$  or  $R_i$ . These commitments are placed on the MO's within the envelopes. When the bank opens the 99 envelopes it asks Alice to reveal the  $L_i$  and  $R_i$  and confirms that they reveal Alice's identity. When Alice asks Bob to produce an identity string he produces a random  $n$  bit vector.

- ▶ If the  $i$ th bit is one Alice reveals  $L_i$ .
- ▶ If the  $i$ th bit is zero Alice reveals  $R_i$ .

The identity string on the money order is the data revealed by Alice.

## RSA Based Blind Signatures

Suppose the bank has the RSA private key  $d$  and the corresponding public key  $(N, e)$ .

Suppose that you wish the bank to sign  $M$ ,

- ▶ but that you don't want the bank to know  $M$ !

Proceed as follows.

- ▶ Choose a **blinding factor**  $B$  at random from  $\mathbb{Z}_N^*$ .
- ▶ Compute  $S = MB^e \pmod{N}$ .
- ▶ Send  $S$  to the bank to sign.
- ▶ The bank computes  $T = S^d \pmod{N}$ .
- ▶ You then compute  $R = T/B \pmod{N}$ .

Then  $R$  is the bank's signature on  $M$  because

$$M^d = \left(\frac{S}{B^e}\right)^d = \frac{S^d}{B^e} = \frac{T}{B} = R \pmod{N}.$$

Now go back to our protocol and look what we have achieved.

Alice remains anonymous to the bank in the sense that where she spends her money cannot be traced.

If Alice cheats her identity is revealed - notice that we do not stop her from cheating.

Bob cannot cheat since if he copies a money order and presents it to the bank he will not get his money.

Alice and Bob cannot collude to defraud the bank.

How about a malicious outsider, Eve.

Eave can cheat!

- Suppose Eve eavesdrops on the communication between Alice and Bob and goes to the bank before Bob does.
- Then, when Bob arrives, he is identified as a cheater!
- Suppose Eve spends Alice's money before Alice can.
- Then Alice will be identified as a cheater.

Hence Alice and Bob need to **protect their data** just as they would do with paper money.

We have seen one protocol for anonymous, but traceable, digital cash.

- There are many others!

Transferability and divisibility can be done as well.

The issue, however, is one of **trust** and not of technology.

- No digital cash scheme will be worth anything until banks, shops and customers are prepared to trust it.

Having now seen how commitment schemes can be used in advanced protocols, we now go on to describe them in more detail.

## Commitment Schemes

One problem with many protocols is that the person who "goes first" may be placed at a disadvantage.

- Think of auctions or an online paper-scissors-stone game.

One way around this problem is for the first party to "commit" to their message without revealing it.

We want a **commitment scheme** with the following properties.

- Binding:** The committer cannot change its mind.
- Concealing:** A party that sees a commitment cannot determine the committed value until the committer wants them to (the **de-commitment** phase).

## Paper-Scissors-Stone Game

Consider the game below using a hash based commitment scheme.

Alice  $\rightarrow$  Bob :  $h = H(\text{paper}||R)$  (where  $R$  is a random value)

Bob  $\rightarrow$  Alice : scissors

Alice  $\rightarrow$  Bob : paper,  $R$

Hence Bob wins since "scissors cuts paper"

Bob can check that Alice committed to **paper** because he can verify that  $h = H(\text{paper}||R)$ .

Alice cannot cheat without finding a collision in the hash function  $H$ . This is computationally hard, hence we say the system is **computationally binding**.

Since  $H$  is many-to-one Bob cannot learn **paper** from  $h$  because, for example, there may be an  $R'$  such that  $h = H(\text{scissors}||R')$ .  $R$  and  $R'$ . Hence we say that the system is **informationally theoretically concealing or perfectly concealing**.

## Commitment Schemes

We would like commitment schemes that are

- perfectly binding** and
- perfectly concealing**.

In other words, no matter how much computational effort Alice or Bob throw at the problem

- Alice cannot change her mind and
- Bob cannot determine the value Alice committed to until she de-commits.

However, it is (provably) impossible to achieve both of these conditionals simultaneously, but we can achieve both separately.

## Scheme 1 : Perfectly Binding

Let  $G$  be a group of prime order  $q$  and let  $g$  be a generator.

To commit to a value  $a \in [0, \dots, q-1]$  Alice computes

$$c = g^a$$

and gives  $c$  to Bob.

To de-commit Alice gives  $a$  to Bob.

This has the following properties.

- Perfectly binding** : Only one value  $a$  in  $[0, \dots, q-1]$  corresponds to the published  $c$ .
- (somewhat!) **Computationally concealing** : Bob can determine  $a$  by solving the DLP in  $G$ .

## Scheme 2 : Perfectly Concealing

Let  $G$  be a group of prime order  $q$  and let  $g$  be a generator. Let  $h \in G$  be such that **neither** Alice **nor** Bob knows the value of  $x$  for which  $h = g^x$ .

For Alice to commit to a value  $a \in [0, \dots, q-1]$  she generate  $b \in [0, \dots, q-1]$  at random and computes

$$c = g^a \cdot h^b$$

which she gives to Bob.

To de-commit she gives Bob  $a$  and  $b$ . This is

- Computationally binding** : Alice can cheat by solving the DLP for  $h = g^x$  to the base  $g$ .
- Perfectly concealing** : Even if Bob can solve the DLP, given  $c$ , for every value of  $a$  there is a value of  $b$  which gives this  $c$ .

This is called the **Pederson commitment scheme**.

## Commitment Schemes

Commitment schemes are a basic component for a number of high level protocols:

- on-line gaming,
- electronic voting,
- e-cash,
- etc.

An interesting property of our two schemes above is that they are **homomorphic** this is a useful property for **secure multi-party computation** because

- one can compute functions of the committed values without knowing what the committed values are!



## ZK-Proofs and Identification

To provide an identification service using a ZK-proof system we equip the user with a password that is the solution to a hard problem.

To login the user runs the ZK-proof protocol  $n$  times.

- ▶ Suppose that an adversary has with probability at most  $p$  of successfully cheating in any one run of the protocol.
- ▶ If for all  $n$  trials the user answers correctly then there is a chance of at most  $p^n$  that it was successful without knowing the password.
- ▶ No amount of eavesdropping will reveal any information about the password!

Almost all interactive ZK-proofs run in the three stages:

- ▶ Peggy commits; Victor challenges; Peggy responds.

Having interactive protocols can be a pain, but we can remove this need using hash functions.

- ▶ Peggy produces  $n$  random isomorphic (to  $G_1$  and  $G_2$ ) graphs and their corresponding isomorphisms.
- ▶ She commits to these graphs by feeding them into a one-way, cryptographic hash function.
- ▶ She then parses the bits of the output of the hash function:
  - ▶ if bit  $k$  is one, she proves that the  $k$ th graph is isomorphic to  $G_1$ ;
  - ▶ if bit  $k$  is zero, she proves that the  $k$ th graph is isomorphic to  $G_2$ .

This is the basic idea behind the Fiat-Shamir heuristic.

Unfortunately, if we replace the challenge with the output of a hash function, we no longer have a ZK-proof.

We cannot simulate!

However, we would be able to simulate if, instead of using a real hash function, we used a random oracle.

By making the hash function depend on a commitment and a message one can a digital signature scheme.

## ZK-Identification and Signatures

Consider the following protocol that Peggy uses to prove that she knows the discrete logarithm  $y = g^x$  in a group  $(G)$  of order  $q$ .

- ▶ Peggy sends a commitment  $r = f(g^x)$  to Victor.
- ▶ Victor sends a challenge  $h$  back to Peggy.
- ▶ Peggy responds with  $s = (h + xr)/k \pmod{q}$ .

Victor verifies that

$$r = f(g^{y^s})$$

where  $u = h/s \pmod{q}$  and  $v = r/s \pmod{q}$ . (By replacing  $h$  with  $H(m)$  we obtain the DSA signature algorithm.)

- ▶ This protocol is complete: If Peggy knows  $x$  then Victor will accept.
- ▶ This protocol is sound with error probability  $1/q$ . (No need to repeat many times to reduce this probability to something very small.)
- ▶ This protocol is zero-knowledge. Generate  $u, v$  at random. Then compute  $r = f(g^{u^k})$ ,  $s = r/v$  and  $h = us$

## ZK-Identification and Signatures

However, the conversion of the protocol into a signature scheme did not follow our general method for making an interactive protocol non-interactive:

- ▶ The challenge *did not* depend on the commitment.

Can we improve on DSA then?

The following is a better protocol.

- ▶ Peggy sends a commitment,  $r = g^h$  to Victor.
- ▶ Victor sends a challenge  $h$  back to Peggy.
- ▶ Peggy responds with  $s = k + hx \pmod{q}$ .

Victor verifies that

$$g^s = r \cdot y^h \pmod{p}.$$

By replacing  $h$  with  $H(m|r)$  we obtain the Schnorr signature scheme.

## Schnorr Signatures

Private Key :  $x$  chosen at random from  $\{1, \dots, q-1\}$

Public Key :  $y = g^x$

Sign ( $m$  using  $x$ ) :

- ▶  $r = g^k$  ( $k$  chosen at random from  $\{1, \dots, q-1\}$ )
- ▶  $h = H(m|r)$
- ▶  $s = k + hx \pmod{q}$
- ▶ Output  $(h, s)$

Verify  $((h, s)$  using  $y$ ) :

- ▶  $r = g^s \cdot y^{-h} \pmod{p}$
- ▶ Accept if and only if  $h = H(m|r)$ .

To simplify things one often replaces  $y$  with  $y^{-1}$ .

## Honest Verifier Zero Knowledge

The Schnorr identification protocol satisfies a definition of ZK known as honest verifier zero knowledge.

Recall how it works.

- ▶ Peggy sends a commitment,  $r = g^k$  to Victor.
- ▶ Victor sends a challenge  $h$  back to Peggy.
- ▶ Peggy responds with  $s = k + hx \pmod{q}$ .
- ▶ Victor verifies that  $g^s = r \cdot y^h \pmod{p}$ .

This is only honest verifier ZK because

- ▶ we assume that Victor always uses a random challenge and the challenge does not depend on the commitment.

## Special Soundness

Suppose we have a three round protocol such as the Schnorr protocol above with transcript  $(r, h, s)$ .

Property : For any  $x$  and any pair of accepting transcripts  $(r, h_0, s_0)$  and  $(r, h_1, s_1)$  where  $h_0 \neq h_1$ , one can efficiently recover  $x$ .

Definition : The above property is called special soundness.

Note that the special soundness implies that a dishonest prover (one that does not actually know  $x$ ) can respond to at most one challenge and so special soundness implies soundness.

## Special Soundness : Example

For the Schnorr protocol above we obtain

- ▶  $r$
- ▶  $h_0$  and  $h_1$
- ▶  $s_0$  and  $s_1$

such that

$$g^{s_0} = r \cdot y^{h_0} \pmod{p} \text{ and } g^{s_1} = r \cdot y^{h_1} \pmod{p}$$

Hence

$$y^{h_0-h_1} = g^{s_0-s_1} \pmod{p}.$$

If  $g$  has order  $q$  then discrete logarithm of  $y$  with base  $g$  is

$$x = (s_0 - s_1)/(h_0 - h_1) \pmod{q}.$$

Almost all **honest verifier zero-knowledge identification protocols** can be converted into **digital signature schemes**.

This can be done by replacing Victor's challenge with the hash of the message to be signed concatenated with the commitment.

This is the **Fiat-Shamir heuristic**.

Suppose we create a signature scheme using the Fiat-Shamir heuristic.

How do we prove that it is secure?

One proof technique relies on the **random oracle model** (ROM).

A **random oracle** is a function that produces a random output on any given input.

- ▶ It always produces the same output when asked the same question again.

This essentially allows us to treat the signature scheme as a **simulatability identification scheme**.

- ▶ In many cases also provides the method of proof of security against an active adversary.

The ROM is useful for many other signature schemes.

The existence of random oracles is unknown (actually nobody believes that they exist).

However, if we assume they exist then we can use them in place of hash functions.

- ▶ This allows us to prove (or argue) that certain schemes are secure.
- ▶ Such proofs are said to be in the **random oracle model** of computing.

When using a scheme in real life we replace the random oracle by a real hash function such as

- ▶ MD5,
- ▶ SHA-1,
- ▶ SHA-2 or
- ▶ RIPEMD-160.

## Random Oracle Model Proofs

We can prove the Schnorr signature scheme secure in the ROM by modelling the hash function  $H$  as a random oracle.

**Taming the Adversary**: Roughly speaking, using the ROM we can argue that the (non-interactive) signature algorithm has the same security properties as the (interactive) identification scheme on which it is based.

Since the underlying identification scheme is a ZK scheme this leads to the adversary not learning anything from being able to query a **signing oracle**.

- ▶ Hence if the scheme is secure against passive adversaries it is secure against active adversaries.
- ▶ This uses the **zero-knowledge simulation** property.

## Random Oracle Model Proofs

To prove that the Schnorr signature scheme is secure we use a technique called the **forking lemma**.

This relies on the ROM.

It uses a notion similar to special soundness.

Essentially we obtain two, related signatures with different hash values on the same message/challenge combination.

This allows us to recover the private signing key which is equivalent to finding a discrete logarithm. And this is believed to be hard.

## Proof Sketch for Schnorr Signatures

We assume we have a forger  $F$  which produces existential forgeries for Schnorr signatures given a public key  $y = g^x$ .

$F$  is a probabilistic polynomial time Turing machine with the following properties.

- ▶ It has a random input tape (for the random choices of  $F$ ).
- ▶ It has access to a hash function oracle  $H$  to which it can make  $q_H$  queries.
- ▶ It has access to a signature oracle  $S$  to which it can make  $q_S$  queries.

As  $F$  is polynomial time,  $q_H$  and  $q_S$  are polynomial in the security parameter.

We assume (without loss of generality) that

- ▶ every signature query uses a hash query ( $q_S \leq q_H$ ) and that
- ▶ the output forged signature is verified by the forger (a **critical hash query** is made by  $F$ ).

## Proof Sketch for Schnorr Signatures

We want to use  $F$  to construct an algorithm  $A$  to find  $x$  in polynomial time.

- ▶ If we assume that one cannot solve DLP in polynomial time then such a forger cannot exist.

Algorithm  $A$  needs to provide  $F$  with its oracles.

- ▶  $A$  needs to answer hash function queries.
- ▶  $A$  needs to answer signature queries **without knowing  $x$** .

## Proof Sketch for Schnorr Signatures

### Hash Queries

$F$  assumed to run in the ROM.

- ▶  $A$  can **simulate**  $H$  provided that its responses have the correct distribution (i.e. are indistinguishable from a RO from  $F$ 's perspective).

$A$  **simulates** the hash function as follows.

- ▶ It keeps a list  $L = \{(h_1, a_1), (h_2, a_2), \dots\}$ .
- ▶ When  $F$  asks  $H$  the query  $a$ 
  - ▶ if  $(h, a) \in L$  for some  $h$  then  $A$  returns  $h$ ,
  - ▶ otherwise  $A$  generates  $h$  at random, inserts  $(h, a)$  into  $L$  and returns  $h$ .

Since  $F$  expects a RO it cannot detect that it is being fooled by the algorithm  $A$  using the above simulation.

## Proof Sketch for Schnorr Signatures

### Signing Queries

We use the fact that the Schnorr ID protocol is honest verifier ZK and therefore **simulatable**.

$A$  simulates a signing query for  $m$  as follows.

- ▶  $A$  generate  $h$  at random.
- ▶  $A$  generate  $s$  at random.
- ▶  $A$  computes  $r = g^s/y^h$ .
- ▶  $A$  sets  $a = m||r$ .
- ▶  $A$  inserts  $(h, a)$  in  $L$  and returns  $(h, s)$ .

$F$  cannot tell it is being fooled!

- ▶ All signature produced by the simulation will verify since  $A$  has control over  $H$ .

## Proof Sketch for Schnorr Signatures

Run the forger  $F$  once with

- ▶ a given random tape and
- ▶ a given simulated random oracle as above.

At the end of its execution it outputs a valid message and signature  $(m, (h, s))$

If this signature is valid (with non-negligible probability) at some point  $F$  must make the **critical hash query**  $m||r$  where  $r = g^a/y^b$ .

Algorithm  $A$  now runs  $F$  again with

- ▶ the same random tape and
- ▶ the same random oracle simulation (up to a point).

Algorithm  $A$  chooses an index  $q_c$  from  $\{1, \dots, q_H\}$  at random; from query  $q_c$  onward it uses a new simulation.

## Proof Sketch for Schnorr Signatures

The runs of the forger “fork” when the hash function changes.

With probability  $1/q_H$  the altered query is the critical one.

If so the forger should output the same signature forgery but with a different hash response.

- ▶ Note that input to the hash function is the same.

Hence our two forger runs produce values of

- ▶  $(m, (h_0, s_0))$
- ▶  $(m, (h_1, s_1))$

such that

$$r = g^{a_0}/y^{b_0} = g^{a_1}/y^{b_1}.$$

## Proof Sketch for Schnorr Signatures

We have

$$g^{a_0}/y^{b_0} = g^{a_1}/y^{b_1}.$$

Hence

$$s_0 - xh_0 = s_1 - xh_1$$

and so

$$x(h_1 - h_0) = s_1 - s_0$$

meaning that

$$x = \frac{s_1 - s_0}{h_1 - h_0}.$$

So, we have solve the DLP with probability  $1/q_H$  using two runs of the forger.

## Proof Sketch for Schnorr Signatures

We have created a polynomial time algorithm which

- ▶ using two runs of the forger  $F$  and
- ▶ simulating the hash function queries

allows us to solve DLP with probability  $1/q_H$  in polynomial time.

The above “proof sketch” can be made totally formal by

- ▶ taking into account the fact that  $F$  is probabilistic and therefore has a **success probability**.

## RSA-FDH

We can use the ROM to prove other signature schemes secure, RSA-FDH for example.

The Fiat-Shamir heuristic does not apply to RSA-FDH.

Suppose that we have a hash function

$$H: \{0,1\}^* \rightarrow (\mathbb{Z}/N\mathbb{Z})^*$$

and the RSA function

$$f: \begin{cases} (\mathbb{Z}/N\mathbb{Z})^* & \rightarrow (\mathbb{Z}/N\mathbb{Z})^* \\ x & \mapsto x^e \pmod{N} \end{cases}$$

The RSA problem is, given  $y = f(x)$  to determine  $x$ . The RSA-FDH signature scheme is to compute

$$s = H(m)^d \pmod{N} \text{ (mod } N = f^{-1}(H(m)).$$

We shall use a forger  $F$  that breaks RSA-FDH to solve the RSA problem.

## Proof Sketch for RSA-FDH

The forger  $F$  requires

- ▶ a random input tape (for the random choices of  $F$ ),
- ▶ oracle access to a hash function oracle  $H$  to which it makes at most  $q_H$  queries, and
- ▶ oracle access to a signer  $S$  to which it makes at most  $q_S$  queries.

As  $F$  is polynomial time,  $q_H$  and  $q_S$  are polynomial functions of security parameter - in this case the number of bits in  $N$ .

Assume (without loss of generality) that

- ▶ every signature query uses a hash query (i.e.  $q_S \leq q_H$ ) and that
- ▶ the output forged signature is verified by the forger (i.e. the **critical hash query** is made by  $F$ ).

$A$  is given an RSA modulus  $N$  and a value  $y$ .

- ▶  $A$  is asked to find  $x$  such that  $y = f(x)$ .

## Proof Sketch for RSA-FDH

**Hash Queries:** Before running  $F$  we select  $t \in \{1, \dots, q_H\}$ .

We simulate the random oracle  $H$  using a list

$$L = \{(h, m, s), \dots\}$$

For the  $i$ th query  $m$  we proceed as follows.

- ▶ If  $i = t$  insert  $(y, m, \perp)$  into  $L$  and return  $y$ .
- ▶ If  $m \in L$ , return the corresponding  $h$  from  $L$ .
- ▶ If  $i \neq t$  and  $m \notin L$ 
  - ▶ choose  $s$  at random from  $(\mathbb{Z}/N\mathbb{Z})^*$ ,
  - ▶ compute  $h = s^e \pmod{N}$ ,
  - ▶ insert  $(h, m, s)$  into  $L$ , and return  $h$ .

Still  $F$ , does not know it is being fooled since it works in the ROM.

## Proof Sketch for RSA-FDH

**Signature Queries:**

If  $F$  asks for a signature on  $m$  then we first call the hash oracle with input  $m$ .

If this results in the  $i$ th such hash oracle query then  $A$  terminates with failure.

Otherwise we return the value  $s$  in the list  $L$  corresponding to  $m$ .

## Proof Sketch for RSA-FDH

With probability  $1/q_H$  the value of  $t$  is the **critical query**.

In this case  $A$  will terminate and output

$$(m, s)$$

where

$$s = f^{-1}(H(m)).$$

But  $H(m) = y$  by construction, so the value of  $s$  that  $A$  returns is the preimage of  $y$  under  $f$ .

So, with probability  $1/q_H$ , we have solved the RSA problem.

Again, the above sketch can be made completely formal.

Proofs of security for IND-CCA2 encryption schemes for are quite complicated.

Most public key schemes are used in a hybrid manner.

- ▶ A public key system is used to encrypt a symmetric key (**key encapsulation mechanism**).
- ▶ The symmetric key is used to encrypt the actual data (**data encapsulation mechanism**).

This is now formalised: the **KEM-DEM** or **hybrid encryption** methodology.

#### Key Encapsulation Mechanism

A KEM is an algorithm which takes as input a public key  $y$  and outputs a pair  $(K, C)$  where

- ▶  $K$  is a key for a symmetric encryption function (e.g. DES or AES) and
- ▶  $C$  is an encapsulation (encryption) of  $K$  using  $y$ .

The inverse, **decapsulation** algorithm takes as input  $(C, x)$  where

- ▶  $C$  is an encapsulation under  $y$  of some key  $K$  - or it should be! - and
- ▶  $x$  is the private key corresponding to  $y$ .

It outputs

- ▶ either  $\perp$  if  $C$  is an invalid encapsulation, or
- ▶  $K$  if  $C$  is an encapsulation of the key  $K$ .

#### Data Encapsulation Mechanism

A DEM is a symmetric algorithm which on input of

- ▶ a message  $M$  and
- ▶ a symmetric key  $K$

outputs an encryption  $E$  of  $M$  using key  $K$ .

The inverse operation takes as input

▶  $(E, K)$

and outputs either

- ▶  $\perp$  if  $E$  is an invalid ciphertext or
- ▶  $M$  if  $E$  is an encryption of  $M$  under the key  $K$ .

## A KEM-DEM Hybrid Cipher

Using the primitives we have been discussing, a KEM-DEM hybrid encryption scheme can then be created as follows.

#### Encryption

- ▶  $(K, C) \leftarrow \text{KEM}(y)$
- ▶  $E \leftarrow \text{DEM}(M, K)$
- ▶ Return  $(C, E)$

#### Decryption

- ▶  $K \leftarrow \text{KEM}^{-1}(C, x)$
- ▶ If  $K = \perp$  then return  $\perp$
- ▶  $M \leftarrow \text{DEM}^{-1}(E, K)$
- ▶ If  $M = \perp$  then return  $\perp$
- ▶ Return  $M$

## DEM : Security Definition

A DEM is secure if no adversary can win the following game.

- ▶ Adversary chooses two messages  $m_0$  and  $m_1$ .
- ▶ Challenger encrypts one of the messages using a random key  $K$  to obtain
  - ▶  $c^* = \text{DEM}(m_b, K)$  -  $m_b$  is known but  $b$  is not.
- ▶ Adversary is given  $c^*$  and an oracle to decrypt ciphertexts under the key  $K$ .
  - ▶ It cannot ask for decryption of the challenge  $c^*$ .
- ▶ Adversary must guess the bit  $b$ .

This is a **semantic security** game.

If the adversary didn't have the decryption oracle then any block cipher **should** meet this definition.

## DEM : Construction

To construct a secure DEM we take

- ▶ a **secure** (under passive attack) block cipher  $E$  and
- ▶ a **secure** (cannot produce a MAC without the corresponding key) MAC function  $\text{MAC}$ .

The function  $\text{DEM}(M, K)$  is then constructed as follows.

- ▶ Split  $K$  into  $k_0$  and  $k_1$ .
- ▶  $c_0 \leftarrow E(M, k_0)$ .
- ▶  $c_1 \leftarrow \text{MAC}(c_0, k_1)$ .
- ▶ Return  $C = (c_0, c_1)$ .

## KEM : Security Definition

A KEM is secure if no adversary can win the following game.

- ▶ The adversary is allowed to query a **decapsulation oracle** with respect to the input public key  $y$ .
- ▶ The challenger computes  $(K_0, C^*) \leftarrow \text{KEM}(y)$  and a random key  $K_1$ .
- ▶ The challenger chooses a bit  $b$  and passes  $(C, K_b)$  to the adversary.
- ▶ The adversary must decide whether  $C^*$  is the encapsulation of  $K_0$  or not.
  - ▶ The adversary may not call its oracle on the challenge encapsulation  $C^*$ .

If we have a KEM secure under this definition and a DEM secure under the definition we gave for DEMs then the resulting KEM-DEM, hybrid encryption function will be **IND-CCA2 secure** (i.e. **semantically secure under adaptive chosen message attack**).

## RSA-KEM

RSA-KEM is a scheme being standardised by ISO.

- ▶ It simpler than RSA-OAEP.
- ▶ Unlike RSA-OAEP it **must** be used in a hybrid KEM-DEM manner.
- ▶ It uses a hash function  $H$ .

Using an RSA public key  $(N, e)$ , RSA-KEM works as follows.

- ▶ Generate  $m \in (\mathbb{Z}/N\mathbb{Z})^*$  at random.
- ▶ Compute  $C = m^e \pmod{N}$ .
- ▶ Compute  $K = H(m)$ .
- ▶ Output  $(K, C)$ .

Using the corresponding secret key  $d$ , decapsulation works as follows.

- ▶ Given  $C$  compute  $m = C^d \pmod{N}$ .
- ▶ Output  $K = H(m)$ .

## RSA-KEM : Observations

Note the following about RSA-KEM.

- ▶ Unlike RSA-OAEP and other public key schemes there is no **ciphertext validity check**.
  - ▶ These are not always necessary for a KEM.
  - ▶ The ciphertext validity check for the whole KEM-DEM ciphertext is within the DEM (i.e., the MAC used in the above construction).
- ▶ RSA-KEM is very simple - as close to textbook RSA as possible.
  - ▶ However, we do not use it to encrypt messages, only to encapsulate keys.
- ▶ The hash function is crucial to the security of the scheme.
  - ▶ In the security proof it is modelled as a **random oracle**.

## RSA-KEM : A Security Proof

The security proof is in the random oracle model; we give a rough outline.

- ▶ The simulator wishes to find the RSA inverse of  $C^*$ .
- ▶ It generates a random key  $K^*$  and passes the adversary  $(C^*, K^*)$ .
- ▶ The simulator keeps a list of calls to the hash function and the decapsulation oracle.
  - ▶ These are cooked-up so that the responses to these calls are consistent.
- ▶ At the end the simulator checks to see if the hash function has been called on an  $m$  such that  $m^e = C^* \pmod{N}$ , if so we know  $m$ .

Since  $H$  is a random oracle the only way the adversary can break the KEM is if it makes the hash function call on the appropriate value of  $m$ .

- ▶ Hence, an adversary against RSA-KEM can be used to solve the RSA problem!

## KEM-DEM : Security Properties

It can be shown that if

- ▶ a KEM is secure and
- ▶ a DEM is secure

then the combined KEM-DEM hybrid scheme is IND-CCA2 secure.

Thus the KEM-DEM approach to public key encryption allows us to build schemes in a modular fashion:

- ▶ design a secure KEM;
- ▶ design a secure DEM;
- ▶ combine them to obtain a secure encryption scheme.

Each component can be designed independently.

## Caution About the Random Oracle Model

We end this section with a cautionary note about the ROM.

The adversaries are assumed to work in the ROM.

- ▶ This is NOT a real computing model!
- In particular an adversary is not allowed to
- ▶ "look inside the box" of the hash function, nor
  - ▶ use some subtle interaction between the precise hash function and the mathematical primitive.

In practice the random oracle is replaced by a concrete hash function that is known by any adversary.

## Caution About ROM

Hence, a ROM proof provides **some evidence** of security and is not a real proof.

It is also a **relative** result in that it assumes some hard problem is indeed hard.

- ▶ If factoring is easy all RSA schemes are insecure, proof or no proof.

However, it is the best tool we have to argue about the security of efficient schemes that can be used in real life.

- ▶ It is thus considered a good design tool and technique.

## More on the ROM

Nobody these days produces a new scheme without (at least) a ROM based proof - you could never publish your idea nor have it included in any standard without one.

Note: One can also apply ROM techniques to

- ▶ identification protocols,
- ▶ key agreement protocols,
- ▶ password protocols,
- ▶ ....

## Non-ROM Proofs

Provably secure schemes can be given which do not rely on the ROM.

- ▶ Often they are less efficient.
- ▶ Sometimes they are not very intuitive.

Of particular interest is the work of Cramer and Shoup in this area (provably secure encryption scheme based on the decision Diffie-Hellman problem).

## Advanced Protocols

We shall now consider

- ▶ other zero-knowledge proofs and
- ▶ 'OR' proofs.

The ZK-Proofs we develop are used in advanced protocols such as

- ▶ auction protocols,
- ▶ voting protocols, and
- ▶ e-cash protocols

## Zero-Knowledge Proofs Again

We wish to introduce ZK-proofs for statements other than knowledge of discrete logarithms or graph isomorphisms.

We will fix a standard notation to describe other protocols.

If we wish to prove knowledge of  $X$  via an honest verifier ZK protocol we need the following.

- ▶ A commitment:  $r = R(X, k)$  ( $k$  is a random nonce.)
- ▶ A challenge:  $h$  (chosen at random)
- ▶ A response:  $s = S(r, h, X, k)$

There must be a **verification algorithm**:

- ▶  $V(r, h, s)$

There must be a **simulator**:

- ▶  $r' = S'(h, s)$

We assume that all  $r$ 's and  $h$ 's and  $s$ 's can be efficiently sampled with the correct distribution.

## Schnorr Protocol

We now present the Schnorr protocol with this notation.

**Purpose** : To prove knowledge of  $x$  such that  $y = g^x$ .

$r = R(x, k)$ :

- ▶  $r = g^k$

$s = S(r, h, x, k)$  (where  $h$  is the random challenge):

- ▶  $s = k + h \cdot x \pmod{q}$

$V(r, h, s)$ :

- ▶ Accept if and only if  $g^s = r \cdot y^h$

$r' = S'(h, s)$

- ▶  $r' = g^{s'} / y^h$

The following protocol was presented as part of an e-cash system; it now has many applications.

Peggy wishes to prove she knows a discrete logarithm  $x$  such that

$$y_1 = g_1^x \text{ and } y_2 = g_2^x.$$

This is a proof of **equality of discrete logarithms**.

Assume  $g_1$  and  $g_2$  generate groups of prime order  $q$ .

$$r = (t_1, t_2) = R(x, k):$$

$$\triangleright t_1 = g_1^k, t_2 = g_2^k$$

$$s = S(r, h, x, k):$$

$$\triangleright s = k - h \cdot x \pmod{q}$$

$$V(r, h, s) \text{ (where } r = (t_1, t_2)\text{):}$$

$\triangleright$  Accept if and only if

$$t_1 = g_1^s \cdot y_1^h \text{ and } t_2 = g_2^s \cdot y_2^h.$$

$$(t'_1, t'_2) = S'(h, s)$$

$$\triangleright t'_1 = g_1^s \cdot y_1^h \text{ and } t'_2 = g_2^s \cdot y_2^h.$$

Note, this is very similar to Schnorr's protocol.

The Chau-Pederson protocol is clearly **honest verifier zero knowledge** as the simulation is indistinguishable from a real transcript.

It is also **complete**.

We need to show that it is sound.

We do this by proving that it has **special soundness**.

Assume two protocol runs with

- $\triangleright$  the same commitment,
- $\triangleright$  different challenges and
- $\triangleright$  valid responses.

We need to show that this reveals the common discrete logarithm  $x$ .

In this case the conclusion that we draw is that a dishonest prover can respond to at most one challenge and so the false acceptance rate is "small".

## Chau-Pederson

Suppose we have two valid protocol runs with transcripts  $(r, t_0, s_0)$  and  $(r, h_1, s_1)$  where  $r = (t_1, t_2)$ .

This gives us

$$t_1 = g_1^{s_0} \cdot y_1^{t_0} \text{ and } t_2 = g_2^{s_0} \cdot y_2^{t_0}$$

and

$$t_1 = g_1^{s_1} \cdot y_1^{h_1} \text{ and } t_2 = g_2^{s_1} \cdot y_2^{h_1}.$$

Which implies

$$g_1^{s_0 - s_1} = y_1^{h_1 - t_0} \text{ and } g_2^{s_0 - s_1} = y_2^{h_1 - t_0}.$$

Therefore  $DLOG_{g_1}(y_1) = DLOG_{g_2}(y_2) = x$  where

$$x = (s_0 - s_1) / (h_1 - t_0) \pmod{q}.$$

## Proving Knowledge of a Committed Value

Suppose that there is a protocol in which Alice is required to commit to a value and that Bob will not proceed until Alice proves that she knows the value committed to.

For the commitment scheme

$$a \longmapsto g^a$$

this is simple: apply Schnorr's (HV) ZK-proof of knowledge of discrete logarithm.

For Pederson commitments

$$a \longmapsto g^a \cdot h^r \text{ (random } b)$$

we need something different.

## Proving Knowledge of a Pederson-Committed Value

To prove knowledge of a Pederson-Committed value we need a protocol to prove knowledge of  $x_1$  and  $x_2$  such that

$$y = g_1^{x_1} \cdot g_2^{x_2}$$

where  $g_1$  and  $g_2$  are of order  $q$ .

The following protocol generalises to a proof of knowledge of  $(x_1, \dots, x_n)$  such that

$$y = \prod_{i=1}^n g_i^{x_i}$$

for  $g_i$  of order  $q$ . (We leave the details as an exercise.)

## Proving Knowledge of a Pederson-Committed Value

$$(t_1, t_2) = R((x_1, x_2), (k_1, k_2)):$$

$$\triangleright t_1 = g_1^{k_1}, t_2 = g_2^{k_2}$$

$$(s_1, s_2) = S((t_1, t_2), h, (x_1, x_2), (k_1, k_2))$$

$$\triangleright s_1 = k_1 + h \cdot x_1 \pmod{q}$$

$$\triangleright s_2 = k_2 + h \cdot x_2 \pmod{q}$$

$$V((t_1, t_2), h, (s_1, s_2))$$

$\triangleright$  Accept if and only if

$$g_1^{s_1} \cdot g_2^{s_2} = y^h \cdot t_1 \cdot t_2.$$

Questions: **completeness**, (**special**) **soundness**, **HK zero-knowledge**?

## OR Proofs

Suppose one wants to prove knowledge of a secret  $X$  or a secret  $Y$

- $\triangleright$  without revealing which we actually know.

Assume that there exist honest verifier ZK-proofs for both  $X$  and  $Y$ .

We combine the two proofs together into one **OR proof**:

- $\triangleright$  a technique used in many protocols.

**Key idea:**

- $\triangleright$  For the secret we know we run the ZK protocol correctly.
- $\triangleright$  For the secret we do not know we run the simulated ZK protocol.
- $\triangleright$  We link the two together by linking the commitments.

## OR Proofs

Let the HV ZK protocol for  $X$  be

- $\triangleright r_1 = R_1(X, k_1)$  (commitment)
- $\triangleright h_1$  (challenge)
- $\triangleright s_1 = S_1(r_1, h_1, X, k_1)$  (response)

With verification algorithm

$$\triangleright V_1(r_1, h_1, s_1).$$

And with simulation

$$\triangleright r'_1 = S'_1(h_1, s_1).$$

## OR Proofs

Let the HV ZK protocol for  $Y$  be

- ▶  $r_2 = R_2(Y, k_2)$  (commitment)
- ▶  $h_2$  (challenge)
- ▶  $s_2 = S_2(r_2, h_2, Y, k_2)$  (response)

With verification algorithm

- ▶  $V_2(r_2, h_2, s_2)$ .

And with simulation

- ▶  $r_2^* = S_2^*(h_2, s_2)$ .

Assume that  $h_1$  and  $h_2$  can be treated as bit strings of the same length.

## OR Proofs

Suppose that Peggy knows  $X$ , but not  $Y$ ; she proceeds as to prove to Victor that she knows  $X$  or  $Y$  as follows.

**Commitment:**

- ▶ Compute  $r_1 = R_1(X, k_1)$  as usual.
- ▶ Choose  $h_2$  and  $s_2$  from the correct distribution.
- ▶ Compute  $r_2^* = S_2^*(h_2, s_2)$ .
- ▶ Output  $(r_1, r_2^*)$ .

**Challenge:**

- ▶ The challenge  $h$  is chosen as usual.

## OR Proofs

Recall: Peggy knows  $X$  and she wants to prove to Victor that she knows  $X$  or  $Y$ .

**Response:**

- ▶ Compute  $h_1 = h \oplus h_2$ . (Peggy chose  $h_2$  above.)
  - ▶ Note: Peggy could not compute  $h_1$  beforehand.
- ▶ Compute  $s_1 = S_1(r_1, h_1, X, k)$ .
- ▶ Output  $(h_1, h_2, s_1, s_2)$ .

**Verification:**

- ▶ Verify that  $h = h_1 \oplus h_2$ .
- ▶ Verify that  $V_1(r_1, h_1, s_1)$ .
- ▶ Verify that  $V_2(r_2^*, h_2, s_2)$ .

To make this non-interactive set  $h = H(r_1, r_2^*)$ .

## OR Proofs

In the case where Peggy knows  $Y$  but not  $X$  the protocol is identical except that we simulate for  $X$  and run the real protocol for  $Y$ .

Since the simulation is indistinguishable (by the HV ZK property), Victor cannot tell difference between a protocol run when Peggy knows  $X$  and run when Peggy knows  $Y$ .

Completeness, soundness and HK ZK properties follow from properties of original protocols.

## OR Proofs: Example

Suppose that Peggy wishes to prove knowledge of  $x_1$  or  $x_2$  such that

$$y_1 = g^{x_1} \text{ and } y_2 = g^{x_2}.$$

We use the Schnorr protocol as the basic building block.

Assume that Peggy knows  $x_1$  but not  $x_2$ .

**Commitment:**

- ▶ Compute  $r_1 = g^{k_1}$  for random  $k_1$ .
- ▶ Choose  $h_2$  and  $s_2$  from the correct distribution.
- ▶  $r_2^* = g^{x_2} / y_2^{s_2}$ .
- ▶ Output  $(r_1, r_2^*)$ .

## OR Proofs: Example

On being given the challenge  $h$  Peggy computes her response.

**Response:**

- ▶ Compute  $h_1 = h \oplus h_2$ .
- ▶ Compute  $s_1 = k_1 + h_1 \cdot x_1 \pmod{q}$ .
- ▶ Output  $(h_1, h_2, s_1, s_2)$ .

**Verification:**

- ▶ Verify that  $h = h_1 \oplus h_2$ .
- ▶ Verify that  $g^{s_1} = r_1 \cdot y_1^{h_1}$ .
- ▶ Verify that  $g^{s_2} = r_2^* \cdot y_2^{s_2}$ .

## OR Proofs

These OR proofs can be extended to an arbitrary number of disjunctions of statements.

Given  $n$  statements of which the prover only knows one secret:

- ▶ Create  $n - 1$  commitments using simulated transcripts for the unknown statements.
- ▶ Commit as usual to the known statement.
- ▶ Generate the correct challenge for the known statement via

$$h = h_1 \oplus \dots \oplus h_n.$$

This can be made non-interactive via the Fiat-Shamir heuristic as usual.