

ECC Protocols

Nigel Smart

nigel@cs.bris.ac.uk

January 27, 2009

Elliptic Curve Based Protocols

Key Agreement

Encryption

Encryption Basics
KEMs and DEMs
EC-IES

Signatures

EC-DSA
Schnorr Signatures
Link With Identification Schemes
Signature Scheme Proofs

Elliptic Curve Based Protocols

Protocols

We shall now move more towards the real world

Concentrating on protocols which are more ECC in flavour

Some therefore may be new to those who have not seen ECC before

- ▶ EC-DH
- ▶ EC-DSA
- ▶ EC-IES
- ▶ EC-MQV

Key Agreement

Elliptic Curve Diffie-Hellman

Easiest to understand of all the protocols.

Two people, Alice and Bob, want to agree a shared secret.

 $E(\mathbb{F}_q)$ is an elliptic curve over a finite field for which ECDLP is hard. P is a point of large prime order.

EC-DH

Alice	Bob
x	$\xrightarrow{[x]P}$ $[x]P$
$[y]P$	$\xleftarrow{[y]P}$ y

Alice can now compute

$$K_A = [x]([y]P) = [xy]P$$

Bob can now compute

$$K_B = [y]([x]P) = [xy]P$$

and

$$K_A = K_B$$

Very small bandwidth if one uses point compression.

EC-DHP

Given

 $[x]P$ and $[y]P$

the problem of recovering

 $[xy]P$

is called the Elliptic Curve Diffie-Hellman Problem (ECDHP).

If we can solve ECDLP then we can solve ECDHP.

It is unknown if the other implication holds.

A proof of equivalence of the DHP and DLP for many black box groups follows from work of Boneh, Maurer and Wolf.

This proof uses elliptic curves in a crucial way.

- ▶ EC-DH is standardised in ANSI X9.63.

EC-DH

In practice life is not so simple, for example in EC-DH we have a man-in-the-middle attack

Alice	Eve	Bob
x	\rightarrow	$[x]P$
$[m]P$	\leftarrow	m
$[xm]P$		$[xm]P$
		n
	\rightarrow	$[n]P$
	\leftarrow	y
		$[yn]P$

Alice agrees a key with Eve, thinking it is Bob Bob agrees a key with Eve, thinking it is Alice

Eve can now examine communications as they pass through her (she acts as a router).

Diffie-Hellman on its own is not enough.

For example how does Alice know who she is agreeing a key with ?

- ▶ Is it Bob or Eve ?

One way around is for

- ▶ Alice to **sign** her message to Bob
- ▶ Bob to **sign** his message to Alice.

In that way both parties know who they are talking to.

Assuming we can construct secure signature schemes we have now solved the key distribution problem:

Authentic public keys are obtained from a CA.

Then secure session keys are obtained using signed Diffie-Hellman

$$\begin{array}{ccc} & \text{Alice} & \text{Bob} \\ ([a]P, \text{Sign}_{\text{Alice}}([a]P)) & \xrightarrow{\quad} & \\ & \xleftarrow{\quad} & ([b]P, \text{Sign}_{\text{Bob}}([b]P)) \end{array}$$

However, it is more common to use the STS-protocol in this situation

When using signed Diffie-Hellman it is common to adopt the [Station-to-Station](#) protocol, or STS protocol

- ▶ $A \rightarrow B : [x]P$
- ▶ $B \rightarrow A : [y]P, \{\text{Sig}_B([y]P, [x]P)\}_{K_{ab}}$
- ▶ $A \rightarrow B : \{\text{Sig}_A([x]P, [y]P)\}_{K_{ab}}$

where $K_{ab} = g^{xy}$

STS provides forward secrecy, but has some subtle problems

Has **unknown key share attack**

- ▶ This attack requires adversary to obtain "invalid" certificates from a CA
- ▶ Very damaging if duplicate signatures can be found

MQV Protocol

System setup

Alice and Bob generate a public/private key pair each

$$(A = [a]P, a) \text{ and } (B = [b]P, b).$$

Via some means (eg a certificate)

- ▶ Bob knows A is authentic
- ▶ Alice knows B is authentic

They now want to agree on a secret session key to which they both contribute a random nonce

- ▶ nonce = number which is used once and then thrown away

Use of the nonce's provides them with forward secrecy

MQV Authenticated Key Exchange

This is the most efficient deployed authenticated key exchange protocol

- ▶ Most analysed authenticated key exchange mechanism available
- ▶ NSA like this one

The **key exchange** Alice and Bob now generate a public/private ephemeral key pair each

$$(C = [c]P, c) \text{ and } (D = [d]P, d).$$

They exchange C and D .

Hence to some extent this looks like a standard Diffie-Hellman exchange with no signing.

However the final session key will also depend on A and B .

MQV

Determining the Session Key

Assume you are Alice

- ▶ You know A, B, C, D, a and c

Let l denote half the bit size of the group G

- ▶ e.g. $l = 160/2 = 80$

Shared secret K computed via

- ▶ Convert C to an integer i
- ▶ Put $s = (i \pmod{2^l}) + 2^l$
- ▶ Convert D to an integer j
- ▶ Put $t = (j \pmod{2^l}) + 2^l$
- ▶ Put $h = c + sa$.
- ▶ Put $K = [h]([t](DB))$

Why does MQV this work ?

Note that s and t seen be Alice, are swapped when seen by Bob,

- ▶ $s_{\text{Alice}} = t_{\text{Bob}}$
- ▶ $t_{\text{Alice}} = s_{\text{Bob}}$

Let

- ▶ h_{Alice} denote the h seen by Alice
- ▶ h_{Bob} denote the h seen by Bob

Then

- ▶ $P = g^{h_{\text{Alice}} \cdot h_{\text{Bob}}}$

You should check this for yourself

MQV however still suffers from a unknown key-share attack

- ▶ But a very subtle attack, hence probably not important in practice

Encryption

Encryption Basics

The "raw" version of RSA or ElGamal in textbooks is a cryptographic **primitive**, real encryption **schemes** used in practice are built from primitives using additional primitives such as hash functions.

- ▶ We combine the **primitives** together into a **scheme**
- ▶ The idea being to avoid problems with using the primitive on its own.

Raw RSA is deterministic: if you encrypt the same plaintext twice you get the same ciphertext.

This is a bad idea, but to see why you need to consider various formal models of security.

Our discussion will be in the context of public key encryption, but similar notions hold for symmetric ciphers.

There are three basic attack models.

- ▶ Passive attack
- ▶ Chosen-ciphertext attack
- ▶ Adaptive chosen-ciphertext attack

This is a very weak form of attack.

Eve is allowed to look at various encrypted messages.

She also has access to the encryption mechanism since we are dealing with a public key scheme.

This models the weakest attack on a public key system.

It is sometimes referred to as **chosen plaintext attack** (CPA).

Chosen-Ciphertext Attack (CCA)

This is often called a **lunch time attack**.

It is slightly stronger than passive attack.

Eve now has access to a black box which performs decryptions (a **decryption oracle**).

Eve can ask the box to decrypt a polynomial number of ciphertexts of her choosing (during the lunch time of the box owner).

After this she is given a target ciphertext and asked to decrypt this on her own.

Adaptive CCA (CCA2)

This is a very strong form of attack.

Eve is given a target ciphertext to decrypt.

She allowed to ask the box to decrypt ciphertexts of her choice.

The only restriction is that she is not allowed to decrypt the target ciphertext using the box.

Security Notions

Along with notions of attack are notions of security:

- ▶ Perfect Secrecy
- ▶ Semantic Security
- ▶ Polynomial Security
- ▶ Non-Malleability
- ▶ Plaintext Awareness

Perfect Security

Recall, a scheme is **perfectly secure** if a passive adversary, with **infinite** computing power can learn nothing about the plaintext given the ciphertext.

This essentially means the key is as long as the message.

- ▶ One time pads are perfectly secure under passive attacks

This is the major theorem of Shannon in the information theoretic area of cryptography.

It is often called **information theoretic security**.

Note one time pad is **not** secure under active attacks.

Semantic Security

This is like perfect secrecy but we only allow an adversary with **polynomially bounded** computing power.

Formally: For all probability distributions on the message space, whatever a passive adversary can compute in polynomial time about the plaintext given the ciphertext, they could also compute without the ciphertext.

i.e. Having the ciphertext does not help one to learn anything about an encrypted message.

Hard to understand but luckily this is equivalent to **polynomial security** which is easy to understand...

Polynomial Security

Suppose that you are given an encryption function f .

You choose two messages m_1 and m_2 .

You are given a ciphertext c such that

$$c = f(m_1) \text{ or } c = f(m_2).$$

A scheme is **polynomially secure** if in polynomial time you cannot decide which message c is the encryption of, with probability significantly greater than 0.5.

Note: This means a polynomially secure encryption function must be non-deterministic.

A scheme that is polynomially secure is often said to have **indistinguishability of encryptions** (IND).

A scheme is non-malleable if given a ciphertext C containing a message M it is impossible to determine a valid ciphertext on a message related to M .

Note: Related is vaguely defined here on purpose.

Raw RSA is malleable owing to the homomorphic property.

This is a VERY STRONG notion of security.

A scheme is called plaintext aware if it is **computationally difficult** to construct a valid ciphertext without knowing the corresponding plaintext to start with.

Plaintext awareness implies one cannot mount a CCA attack.

- ▶ To write down a ciphertext requires you to know the plaintext.
- ▶ So why would you ask the decryption oracle to decrypt the ciphertext ?

Combined with notions of attacks and notions of security are notions of computing models.

The standard model is what one usually works with in computing.

There is another model used in cryptography: the **random oracle model**.

In this model one assumes that idealised hash functions exists.

Cryptographic Hash Functions

A **cryptographic hash function** h is a function that

- ▶ takes bit strings of arbitrary finite length as input and
- ▶ maps them to bit strings of a fixed length (l below).

We write

$$h : \{0, 1\}^* \rightarrow \{0, 1\}^l.$$

The output of h on string s is referred to as the **hash code**, the **hash value**, the **hash**, the **digest** or the **fingerprint** of s .

Cryptographic Hash Functions

A cryptographic hash function should have the following properties.

Preimage Resistant : Hard to find a message with a given hash value.

Collision Resistant : Hard to find two messages with the same hash value.

Second Preimage Resistant : Given $h = h(m)$ it is hard to find m' with $h(m') = h(m)$.

Random Oracle Model

We assume the existence of **random oracles** functions that

- ▶ take polynomial time to evaluate and
- ▶ to an observer cannot be distinguished from random functions.

The existence of such a function would probably prove $P \neq NP$.

We prove a protocol secure in the random oracle model and then make a real life protocol by replacing the random oracle with a hash function like SHA-1 or MD5.

ElGamal - Malleability

Recall that a finite field based ElGamal ciphertext is of the form

- ▶ $(kP, M + kQ)$

where

- ▶ k is an ephemeral per-message secret and
- ▶ Q is the public key.

This is malleable; on seeing this ciphertext Eve can create a valid ciphertext of the message $M + P$ without knowing M .

- ▶ $(kP, P + M + kQ)$

Chosen Ciphertext Security

Malleability implies that the scheme is not secure against adaptive chosen ciphertext attack.

Suppose that Eve wants to decrypt a target ciphertext c .

She is allowed access to a decryption oracle with any ciphertext other than c itself.

What does she do to decrypt c ?

- ▶ She "alters" c to create a new ciphertext c' .
- ▶ She obtains the decryption m' of c' from her decryption oracle.
- ▶ Finally, she recovers m from m' .

CCA - ElGamal

The ciphertext that Eve wants to decrypt is

$$c = (C_1, C_2) = (kP, M + k \cdot Q).$$

where $Q = xP$ is the public key and x is the private key.

Eve creates

$$c' = (C_1, P + C_2) = (C_1, C_2')$$

Eve asks her decryption oracle to decrypt c' ; the oracle returns M' .

Eve then computes

$$\begin{aligned} M' - P &= (C_2' - x \cdot C_1) - P \\ &= (P + C_2) - xk \cdot P - P \\ &= M + k \cdot Q - xk \cdot P \\ &= M + kx \cdot P - xk \cdot P = M. \end{aligned}$$

However "raw" ElGamal is quite good under CPA attacks

Assuming the DDH assumption then ElGamal is IND-CPA

Notice the difference with RSA, under CPA attacks raw deterministic RSA cannot be IND secure at all.

Also raw ElGamal requires messages to be encoded as group elements.

- ▶ Which can be a pain for elliptic curve groups

KEMs and DEMs

Before introducing ECIES we recap on the modern method for creating public key encryption schemes.

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 Mechanisms

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 Mechanisms

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 : 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 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-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.

EC-IES

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.

This is the standard ECC encryption algorithm.

- ▶ Based on Abdalla, Bellare and Rogaway's DHAES protocol
- ▶ Secure under a non-standard assumption (Abdalla et. al.)
- ▶ Secure under the ROM (Abdalla et. al.)
- ▶ Secure under the Generic Group Model (Smart)

IES stands for integrated encryption scheme

- ▶ The scheme works like static Diffie-Hellman followed by symmetric encryption.
- ▶ We use Diffie-Hellman as a KEM
- ▶ Then encrypt the message with a DEM

A symmetric encryption scheme $\text{SYM} = (E_k, D_k)$,

- ▶ Key space K_1

A MAC function MAC_k

- ▶ Key space K_2 .

A key derivation function V .

- ▶ The key derivation function V will map group elements
- ▶ to the key space of both the encryption and MAC functions.

The scheme ECIES is defined as a triple of randomised algorithms,

- ▶ **{keygen, enc, dec}**.

- ▶ $d \leftarrow \{1, \dots, q\}$.
- ▶ $Q \leftarrow [d]P$.
- ▶ Return (Q, d) .

EC-IES Encryption

- ▶ $k \leftarrow \{1, \dots, q\}$.
- ▶ $U \leftarrow [k]P$.
- ▶ $T \leftarrow [k]Q$.
- ▶ $(k_1, k_2) \leftarrow V(T)$
- ▶ $c \leftarrow E_{k_1}(m)$
- ▶ $r \leftarrow \text{MAC}_{k_2}(c)$
- ▶ Return $e \leftarrow U||c||r$.

The cipher text is (U, c, r) .

- ▶ U is needed to agree a key
- ▶ c is the actual encrypted message
- ▶ r is used to avoid adaptive chosen ciphertext attacks

The data item U can be compressed to reduce bandwidth.

EC-IES Decryption

- ▶ Parse e as $U||c||r$
- ▶ $T \leftarrow [d]U$
- ▶ $(k_1, k_2) \leftarrow V(T)$
- ▶ If $r \neq \text{MAC}_{k_2}(c)$
 - ▶ Return **invalid**
- ▶ $m \leftarrow D_{k_1}(c)$.
- ▶ Return m .

ECIES

EC-IES makes it easy to encrypt long messages

Standardized in a number of places

- ▶ ANSI X9.63
- ▶ IEEE P1363
- ▶ SEC 1
- ▶ etc

EC-IES Practical Considerations

ECIES (as defined in ANSI X9.63 and SEC-1) is different from DHIES/DHAES

- ▶ V is only applied to the group element T .

The original DHAES paper suggested using a key derivation function of the form $V(U, T)$.

- ▶ Otherwise this can lead to trivial malleability of the ciphertext in the case when a group of non-prime order is used.
- ▶ This is a major problem in finite field systems.
 - ▶ Not such a problem in elliptic curve systems.

EC-IES Practical Considerations

Some standards specify using

- ▶ $V = x - \text{coord}(P)$

as the key derivation function.

This leads to a trivial malleability since we have the collision

- ▶ $V(P) = V(-P)$.

Solution, could use the bit representation of the compression of the point P .

EC-IES Practical Considerations

Assume we have a ciphertext of the form

- ▶ $c = U||c||r$

corresponding to the plaintext m , where U is of prime order q but the underlying group is of order $2q$.

If the private key d is even then trivially obtain another valid ciphertext, corresponding to the plaintext m ,

$$(U + H)||c||r$$

where H is an element of order two.

Since

$$V([d]U) = V([d](U + H)).$$

To avoid this trivial malleability we need to either

- ▶ use the key derivation function $V(U, T)$

or

- ▶ Use a modified function V , see the ISO standard.
 - ▶ Efficient and helps avoids the "key validation" Certicom patent.

or

- ▶ check that for each received ciphertext, U is an element of order q .
 - ▶ Which is the subject of key validation

Signatures

Digital Signatures

Another very important public key primitive is the **digital signature**.

The idea is

Message + Alice's Private Key = Signature

Message + Signature + Alice's Public Key = YES/NO

Alice can **sign** a message using her private key.

Anyone can **verify** Alice's signature, since everyone can obtain her public key.

After verification the verifier is convinced that only Alice could have produced the signature because

- ▶ only Alice knows her private key!

Note: The above outline is a **signature scheme with appendix**.

Digital Signatures

On the last slide we described a signature scheme **with appendix**: the message is an explicit input of the verification algorithm.

Some signature schemes have the property of **message recovery**: the message is recovered from a signature.

The basic idea is

Message + Alice's Private Key = Signature

Signature + Alice's Public Key = Message or **INVALID**

Henceforth we denote a public/secret key pair (pk, sk) .

A message is denoted m , the signing algorithm is denoted S , the verification algorithm is denoted V a signature is denoted s

So, we have $S(sk, m) = s$ and $V(s, m, pk) = \text{YES/NO}$. (For with appendix case.)

Signature Scheme Security

What does it mean for a signature scheme to be secure?

- ▶ **Note:** We don't care about recovering a message, since the message is public.

Like standard signatures we worry about forgery.

- ▶ There are two types of forgery:
 - ▶ **selective forgery** and
 - ▶ **existential forgery**.

Signature Scheme Security

Selective Forgery :

- ▶ Clearly we require that an attacker should not be able to produce a message, signature pair on a message of their choice.
- ▶ This is considered to be a weak notion of security.

Existential Forgery :

- ▶ A scheme is **existentially unforgeable** if, no matter how many message, signature pairs an adversary sees, it cannot produce a signature on **any** other message.

Signature Scheme Security

We have just seen some **adversarial goals**; to form a complete definition we also need **attack models**.

Passive Attack :

- ▶ Attacker obtains a public key and some message, signature pairs produced using the public key.

Adaptive Chosen Message Attack :

- ▶ Attacker can obtain signatures on messages of its choosing.
- ▶ It can choose the messages based on what it has already seen - hence **adaptive**.
- ▶ It's job is to produce a signature on a new message.

Accepted definition of security :

- ▶ A signature scheme is deemed secure if it resists **existential forgery** under an **adaptive chosen message attack**.

Recall, the basic idea is

Message + Alice's Public Key = CipherText

CipherText + Alice's Private Key = Message

Hence, anyone with Alice's public key can send Alice a secret message.

Only Alice can decrypt the message since

- ▶ only Alice has the private key!

All they need do is look up Alice's public key in some **directory**.

Or receive her public key and a corresponding **certificate**.

Variant of the American DSA algorithm as specified in NIST FIPS 186.

- FIPS 186.2 contains the new version including EC-DSA,
 ▶ This is also in ANSI X9.62, IEEE P1363 and SEC2

With all digital signature algorithms one actually signs a **hash** of the message, M .

- ▶ For ECC this hash function is always $SHA - 1$.
 - ▶ $SHA - 1$ takes an arbitrary length input and produces a 160 bit output.
 - ▶ We interpret this output bit string as a number.
- We also interpret the x -coordinate of a point as a number.
- ▶ Even when $K = \mathbb{F}_{2^m}$.

E an elliptic curve over $K = \mathbb{F}_p$.
 P a point of large prime order, q .

$$\#E(K) = hq$$

h is called the cofactor.

The set $\{K, E, q, h, P\}$ is called the **domain parameters**.

Private Key : $d \in_{\mathbb{R}} [1, \dots, q - 1]$. **Public Key** : $Q = [d]P$.

Choose $k \in_{\mathbb{R}} [1, \dots, q - 1]$.
 Compute $[k]P = (x, y)$.
 Convert x to an integer, r , mod q .
 If $r \equiv 0$ then goto beginning.
 Put $e = SHA - 1(M)$.
 Compute

$$s \equiv (e + dr)/k \pmod{q}$$
.

If $s \equiv 0$ then goto beginning.
 Return (r, s) as the signature.

EC-DSA : Verifying

Put $e = SHA - 1(M)$.
 Reject if $r, s \notin [1, \dots, q - 1]$.
 Compute

$$\begin{aligned} u_1 &\equiv e/s \pmod{q} \\ u_2 &\equiv r/s \pmod{q} \end{aligned}$$

Set $R = (x, y) = [u_1]P + [u_2]Q$.
 Reject if R is at infinity.
 Convert x to an integer, l , modulo q .
 Accept if and only if $r \equiv l$.

EC-DSA

Need to make sure ephemeral exponent is truly random.

Signing is much faster than RSA
 ▶ Verification is slower.

Size of signature is much smaller than RSA.

Scales much better than RSA or DSA over time.

Schnorr Signatures

Schnorr Signatures

- An important DLP based signature scheme is that of Schnorr.
- ▶ It occurs in many ZK proofs and as parts of other protocols.
 - ▶ It is the simplest among the DLP based schemes that are **provably secure**.
 - ▶ The signing and verifying operations are simpler than those for DSA.
 - ▶ Is the basis for many other protocols.
 - ▶ Currently being proposed for standardisation in ISO

Each user generates a secret signing key x at random and such that

- ▶ $0 < x < q$.

Public key is $Q = xP$.

Are you noticing a pattern here?

Schnorr Signatures : Signing

To sign a message M the signer proceeds as follows.

- ▶ Signer chooses a random **ephemeral key**: $0 < k < q$.
- ▶ Signer computes $R = kP$.
- ▶ Signer computes one-way hash $m = H(R||M)$.
- ▶ Finally, signer computes

$$s = (k + mx) \pmod{q}$$

The signature on M is the pair (m, s) .

Schnorr Signatures : Verification

To verify a signature (m, s) on a message M under public key Q , the verifier proceeds as follows.

The verifier computes

$$R' = sP - mQ$$
.

If the signature is valid we have

$$R' = (k + mx)P - xmP = kP$$

So, the verifier accepts signature if and only if

$$m = H(R'||M)$$

Identification is a very general problem in information security.

- ▶ How does one check that a message comes from a particular source and has not been tampered with?

One example is remote login.

- ▶ How can we ensure that only authorised user have access to a resource?

One approach is to use **zero-knowledge proofs**.

The idea is to prove that you know something, for example a password, without actually revealing **any information** about that something.

The following is a common protocol in telephone banking, which is not zero knowledge for obvious reasons.

Alice chooses a password, say "Looking Glass", the bank also keeps a copy of the password.

Alice rings up the bank.

The bank says "What is the 3rd and 9th letter of your password?"

Alice replies "O and L".

The bank says "OK What can we do for you today?"

After enough runs of this protocol an eavesdropper would know all of Alice's password!

[Link With Identification Schemes](#)

Security Properties for ZK-Proofs

A ZK-Proof should have the following security properties.

Completeness : If the prover **knows** the secret then the verifier accepts.

Soundness : If the prover does **not know** the secret then the verifier accepts with "small" probability.

- ▶ By repeating the protocol we can make the false acceptance arbitrarily small.

Zero-Knowledge : The verifier learns nothing about the secret.

- ▶ In particular, the verifier cannot use the transcript of the protocol to convince someone else they know the secret.

Zero-Knowledge Protocols

Real zero-knowledge protocols are based on knowing the solution to some hard problem.

For example the **graph isomorphism problem**.

- ▶ A graph isomorphism is a permutation of the vertices of one graph to obtain another **isomorphic** graph.

Graph G_1 :

- ▶ Vertices : $V = \{1, 2, 3, 4\}$
- ▶ Edges : $E = \{12, 13, 14, 34\}$

Graph G_2 :

- ▶ Vertices : $V = \{1, 2, 3, 4\}$
- ▶ Edges : $E = \{12, 13, 23, 24\}$

The permutation linking these two graphs is $\sigma = (1, 2, 4, 3)$.

- ▶ i.e. $1 \rightarrow 2, 2 \rightarrow 4, 4 \rightarrow 3, 3 \rightarrow 1$.

Graph Isomorphism Protocol

Peggy has graphs G_1 and G_2 and has a secret **isomorphism** between them.

- ▶ In other words, the vertices of G_2 are a known - to Peggy - relabelling of the vertices of G_1 .

When challenged by Victor, Peggy generates a new graph H by a random permutation and the isomorphism between G_1 and G_2 .

- ▶ Peggy gives H to Victor.
- ▶ Victor asks for either
 - ▶ an isomorphism between G_1 and H , or
 - ▶ an isomorphism between G_2 and H .
- ▶ Peggy complies by sending Victor the required isomorphism.

Graph Isomorphism Protocol

The protocol is clearly **complete**:

- ▶ If Peggy knows the isomorphism then Victor will accept.

The protocol is also **sound**, with error probability 0.5 .

The main issue is whether it is **zero knowledge**.

- ▶ To establish this we must introduce the notion of **simulatability**.

Simulatability

Why do we say the above is a ZK-Proof?

We need to demonstrate that no one who eavesdrops on the protocol, or even Victor himself, can learn anything from the protocol other than the fact that Peggy knows an isomorphism.

This will clearly hold if Victor could write down a transcript of the protocol without Peggy being involved.

- ▶ After all, if Victor could write down a transcript without Peggy, then Victor cannot use a valid transcript to convince another party that he knows something about Peggy's secret.

This ability to write down a transcript of the protocol without knowledge of the secret means that the protocol is **simulatable**.

- ▶ It essentially means that Victor being convinced of Peggy's knowledge comes from the **interactive** nature of the protocol.

Simulatability

We can simulate the protocol transcript as follows.

- ▶ First choosing G_1 or G_2 , say G_0 .
- ▶ Then generating H so that we know the isomorphism between H and G_0 .

We can then publish the transcript containing

- ▶ H ,
- ▶ G_0 and
- ▶ the isomorphism.

Notice in the above we have written these things in the order that they are generated in a valid run of the protocol. However, for the simulation, we generated them in the order

- ▶ G_0 ,
- ▶ the isomorphism, and finally
- ▶ H .

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 has 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 $r = f(g^k)$ 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 $v = r/s \pmod{q}$.

- ▶ 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 $r = f(g^{y^u})$, $s = r/v$ and $h = us$

ZK-Identification and Signatures

In the previous identification scheme if we replace h with $H(m)$ we obtain the DSA signature algorithm.

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?

- ▶ We could use a better basic identification protocol
- ▶ We could then apply the Fiat-Shamir heuristic correctly.

ZK-Identification and Signatures

The following is a better protocol, (the Schnorr identification protocol).

- ▶ 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

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

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

- ▶ h depends on the message **and** the commitment.

Signature Scheme Proofs

Security Summary

An adversary runs allowed to ask for a polynomial number of signatures of his choosing

At end adversary must output a signature on any message for which they have not queried their signature oracle.

- ▶ EC-DSA is proved secure in the generic group model (Brown) but not in the ROM
- ▶ DSA: No known proof in any model
- ▶ Schnorr : Proof in both ROM (Pointcheval, Stern)
- ▶ EC-Schnorr : Proof in the GGM (Neven et. al.)

Caution About the Random Oracle Model

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

- ▶ Similar comments apply about the Generic Group Model!

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.

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.

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,
- ▶

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).