PK issues, ideas

- RSA *cipher*

- Diffie-Hellman *key exchange*

- Authentication issues

- Examples of protocols
  - * Threshold schemes
  - * Oblivious transfer
  - * Zero-knowledge proofs

- e-money and identity issues

Authentication issues

Possibly *authentication* is even more important than *secrecy*.

Consider *reliability* of communication in a *hostile* (not merely *noisy*) environment.

Obstacles to communication can be more serious than loss of confidentiality.

Both public-key and private-key techniques are used to address (with varying success) these issues.

Message authentication would confirm/assure

- Identity of sender
- Origin of messages
- Content of messages
- Sequence of messages
- Timing of messages
- Receipt of messages
- Non-repudiation of messages

Identity of sender: One would not want attackers to be able to send messages to your correspondents *impersonating* you. And you would not want third parties to be able to pose as one of your trusted correspondents.

Origin: The issue of physical location or *host* is distinct from *identity*, but is entangled with it, since often the actions of your computer are by default attributed to you. As an important example of the hazard of spoofing: in many email systems it is easy to send mail in the local network with whatever return address one wishes, thus appearing to originate with the corresponding person. A number of email viruses do this.

Content: Even if the content of a message need not be secret, you would not want an adversary to be able to *alter* the content.

Sequence: You would not want an adversary to delete some of a sequence of messages, or reorder a sequence of messages.

Timing: An adversary should not be able to **replay** one or more old messages as if they were new (**replay attack**), or be able to **delay** messages to make them appear that they were sent later than they actually were.

Receipt: You would not want an adversary to be able to falsely report that a message was lost, or to report falsely that it was received.

Non-repudiation: You would not want a sender to be able to deny that they sent a message to you. One may not entirely trust one's correspondents.

In all cases there are two possibilities: **prevention** of an attack, and **detection** of the attack (with corresponding adaptation).

Prevention would be ideal, but for these issues at the present time (lacking easily available quantum channels) it seems that *detection* is a practical goal.

Given the whimsicalness of networks, messages can get out of order, be delayed, or be lost *not* due to hostile acts, but random events. Thus, verifying sequence, timing, and receipt is more complicated than verifying origin and content.

There are three somewhat different types of mechanisms used in authentication:
- **message encryption**
- **message authentication codes** (MACs)
- **hash functions**

Message encryption:

Keep in mind that the goal may not be just *secrecy*, but *message integrity*.

In this approach, one encrypts the message using a cipher that the authorized recipient can decrypt, and sends the ciphertext.

The authorized recipient decrypts the ciphertext.

The ciphertext is the **authenticator** of itself, because we presume that no unauthorized party could create a message that would decrypt properly with that cipher/key.

Messages can be stolen, replayed, etc.

Message authentication codes: These are many-to-one functions producing ‘small’ pieces of authentication material by using a public function of the message and of a secret key shared by the authorized recipient.

The small piece of data, often called *the MAC*, is sent along with the plaintext.

The *authenticator object* is the value of the MAC. The recipient of the message recomputes the MAC of the received message and checks that it matches the MAC value sent along with it.

This does not prevent reordering, delaying, replay, or theft of packets.

Hash functions These are message authentication codes without keys.

Just as MACs, hash functions produce a ‘small’ piece of authenticating data by using a public function of the message.

There are situations in which it would not be possible (or would be inconvenient) to have arranged a shared secret key.

Using **encrypted message as authenticator** is tricky with unstructured messages: decryption gives a plaintext whose *legitimacy* is hard to confirm.

A modification: the sender can append a **checksum** (also called **frame check sequence**, or FCS) of the whole message to itself before encryption.

The authorized decryptor decrypts the whole, removes the checksum from the tail of the plaintext, and computes the checksum value of the remaining plaintext to see whether it matches the checksum.

We imagine it would be difficult for an interloper to generate a fake ciphertext which would decrypt to plaintext-plus-checksum. But *replay attacks* are not prevented.

In any case it is essential that the checksum be added **before** encryption.

Encryption does not prevent *replay* of old messages which had been overheard and copied by an eavesdropper. To avoid having old messages be *replayed*, the time of generation of a message should be included as a part of the message itself, as a **timestamp**.

The timestamp should be added to the message before any other authentication code of the message is computed, and before encryption.

Not encrypting but computing a keyed MAC and appending the MAC is computationally cheaper than encrypting, and allows the recipient a **choice** of *whether or not to verify*, based on circumstances. The *secret-key* material used in computing the MAC prevents unauthorized interceptors from altering the message and creating a matching MAC value.

While it sounds good, **verification of receipt** of messages creates difficulties, some possibly unexpected, such as **information leakage**.

If an email receipt protocol were implemented, by sending (uninvited) email one could obtain information about other people without their permission.

The ‘finger’ protocol already allows something of this sort, with the pursuant privacy problems, even if one is not allowed to check last login times. For this reason ‘finger’ is often disabled.

Some aspects of chat rooms and instand messaging cause similar information leakage.

Calling someone to see whether they’re home or in their office, without identifying oneself or deceitfully identifying oneself, is a similar information-theft technique.

Finally, **non-repudiation** is hard to assure. Someone can send you an encrypted message, but later *claim that their key had been compromised prior* to that time, so they are not responsible for the content of the message.

There seems to be no simple solution to this. It can really happen that a key is compromised, and **revocation** must be possible.

This is related to issues about whether one is responsible for the actions of one's computer, whether a claim that the computer had been 'hijacked' is a legitimate defense, and *how* one proves that one did *not* have control of the computer.

This becomes awkward especially if one has specialized technical knowledge.

Examples of protocols

A **protocol** is a formal procedure, to be followed by people or machines, embodying public-key and other computations, to achieve certain pre-specified effects.

The simplest and best-known protocols are implementations of ciphers, with the goal of achieving *secrecy*. Using the RSA *cipher* in a specified manner, or using Diffie-Hellman *key exchange* to exchange an AES key are such.

There are other possibilities, some of which will be important in the future, such as

* Threshold schemes
* Oblivious transfer
* Zero-knowledge proofs

We will not give formal specifications, but only informal sketches.

Threshold schemes

A **threshold scheme** is a mechanism by which which a secret can be uncovered if *sufficiently many*, but not necessarily *all*, members of a group agree.

More precisely: given a **secret** x to be k-**shared** among t people A_1, $A_2, \ldots, A_t$, give A_i a blob of information a_i so that
• A_i knows a_i (but not a_j for $j \neq i$).
• *No part* of the secret x can be recovered from any $k-1$ of the blobs a_i.
• The secret x *can* easily be computed from any k of the a_i's.

That is, the t entities involved **share the secret** in that at least k of them must cooperate to recover the secret. Given t and a secret x, a list of $a_1, \ldots, a_n$ which accomplish these objectives is a (k, t)-**threshold scheme**.

A simple threshold scheme uses Sun Ze's theorem.

Let $m_1, \ldots, m_t$ be pairwise relatively prime integers ≥ 1. Let Let $M = a_1 \ldots a_t$, $M_i = M/m_i$, and $n_i = M_i^{-1} \bmod m_i$.

(Since m_i is relatively prime to m_j for $j \neq i$, m_i is also relatively prime to M_i, by unique factorization, since M_i is the product of integers prime to m_i. Thus, the multiplicative inverse N_i exists and is computable via Euclid's algorithm.)

By Sun Ze, for integers $a_1, \ldots, a_t$ the system of simultaneous congruences

$$x = a_i \bmod m_i \quad \text{for all } i$$

is equivalent to

$$x = \sum_{i=1}^{t} a_i \, M_i \, n_i \mod M$$

Fix k with $1 < k \leq t$. Let H_k be the *smallest* product of k different m_is, and h_{k-1} the *largest* product of $k-1$ different m_is. We assume that H_k is much larger than h_{k-1}:

$$H_k \geq (N+1) \cdot h_{k-1}$$

for positive N.

Theorem: For **secret** x represented as a number in the range

$$h_k < x < H_k$$

let $a_i = x \,\%\, m_i$. Then the set $\{a_1, \ldots, a_t\}$ is a (k, t) threshold scheme for x.

Remark: As can be seen in the proof, this is *more secure* if the m_i are large and close together.

Proof: Suppose $a_1, \ldots, a_k$ are known. Let $M' = m_1 \ldots m_k$, and $M'_i = M'/m_i$ for

$1 \le i \le k$. Let $n_i' = {M_i'}^{-1} \bmod m_i$ for $1 \le i \le k$. Let

$$x' = \sum_{i=1}^{k} a_i \, M_i' \, n_i \mod M'$$

Then (Sun Ze)

$$x' = x \bmod M'$$

Since $M' \ge H_k > x$, secret x is already reduced modulo M', so x can be computed by

$$x = x' \,\%\, M'$$

On the other hand, suppose only $a_1, \ldots, a_{k-1}$ are known. Let $M' = a_1 \ldots a_{k-1}$ and $M'_i = M'/m_i$ for $1 \leq i \leq k-1$. Let $n'_i = {M'_i}^{-1} \bmod m_i$ for $1 \leq i \leq k-1$. Let

$$x' = \sum_{i=1}^{k-1} a_i \, M'_i \, n_i \mod M'$$

Since $M' = m_1 \ldots m_{k-1}$,

$$x' = x \bmod m_1 m_2 \ldots m_{k-1}$$

And $m_1 m_2 \ldots m_{k-1} \leq h_{k-1}$. Since $h_{k-1} < x < H_k$ and since (by hypothesis)

$$(H_k - h_{k-1})/h_{k-1} > N$$

there are at least N possibilities for $y \bmod M$, so that

$$x' = y \bmod M'$$

Thus, knowing only $k - 1$ of the a_i's is insufficient to discover the secret x.

///

To set up a situation in which to use a threshold scheme, we might suppose that there is a **trusted central agency** which has the secret x, which knows the moduli m_i, which computes the threshold scheme data $a_1, \ldots, a_n$, and which communicates a_i to the entity A_i by secure means.

What *remains* is to set up a protocol for k among the t entities to share their information without anyone cheating, or at least without cheating going undetected.

Cheating is a non-trivial problem in design of protocols. **Prevention** is often impossible. **Detection** is easier, but mere detection of damage may be insufficient. One must have adequate plans for **recovery** from damage.

Oblivious transfer

A simple version: Alice has a secret which she wishes to communicate to Bob so that, afterward, Alice herself does not know whether Bob actually received the secret, but Bob knows.

A better version: Alice has *several* secrets, and wishes to transfer one of them to Bob in such manner that only Bob knows which secret was actually communicated.

This could be relevant if Bob does not want to embarrass himself by directly confessing that he is ignorant of one or more of the secrets.

For the simpler case, assume the single secret is the factorization $n = pq$ with two large primes p, q with $p = 3 \bmod 4$ and $q = 3 \bmod 4$.

(This is actually very general, because any other secret could be encrypted via RSA using modulus n, so that knowing the factorization allows decryption, revealing the secret.)

Use the fact that *knowing* $x, y \bmod n$ *so that* $x^2 = y^2 \bmod n$ *but* $x \neq \pm y$ *allows factorization of* n, since $\gcd(x - y, n)$ will be a proper factor of n, exactly as in factoring n by using a square-root oracle.

Bob chooses random $0 < x < n$, sends $z = x^2 \bmod n$ to Alice.

Alice computes principal square roots w_1 of $z \bmod p$ and w_2 of $z \bmod q$ via

$$\begin{aligned} w_1 &= z^{(p+1)/4} \bmod p \\ w_2 &= z^{(q+1)/4} \bmod q \end{aligned}$$

Choosing random $\pm$'s, Alice lets $y_1 = \pm w_1$, $y_2 = \pm w_2$, and uses Sun Ze (and Euclid) to make a square root y of z modulo $n = pq$ with $y = y_1 \bmod p$ and $y = y_2 \bmod q$. Alice sends y to Bob.

If Bob's x was $x = \pm y \bmod n$ Bob cannot factor n, but if

$$\begin{gathered} x = y_1 \bmod p \quad \text{and} \quad x = -y_2 \bmod q \\ \text{or} \\ x = -y_1 \bmod p \quad \text{and} \quad x = y_2 \bmod q \end{gathered}$$

Bob can compute $\gcd(n, x - y)$ to find one of p, q.

Recall that $y^2 = z \bmod pq$ with distinct primes p, q has at most 4 solutions. Thus, in Bob's situation, the two solutions $\pm x \bmod n$ are 2 of the 4 total, and Alice has equal probability of delivering any one of the 4. Thus, after Alice's information, Bob has 50% chance of being able to factor n, and 50% chance that he will *not* be able to factor n.

That is, after a single transaction Alice only knows **probabilities** *for Bob knowing or not knowing, but Bob either does or does not know. Only Bob knows whether he knows or not. Alice does not know whether Bob knows or not.*

Now suppose Alice has two secrets and wants to transfer them to Bob so that Bob gets just one of them, but so that Alice does not know which of the two he got. We assume intractability of computation of *discrete logs* modulo large prime p.

Given b mod p (not necessarily a primitive root), the **discrete log** or **index** $\mathrm{ind}_b(x)$ of x modulo p base b is the non-negative integer ℓ (if it exists) such that

$$b^\ell = x \bmod p$$

If b is a *primitive root* mod p then (as we've seen) the discrete log does exist.

Note that from p, primitive root b mod p, and values $b^x \bmod p$ and $b^y \bmod p$, the value $b^{xy} \bmod p$ *cannot* be easily computed. It seems to require knowing the discrete logs x and y.

The secrets are integers s_1, s_2, padded if necessary to have the same length.

Large prime p is chosen randomly and publicly, primitive root b for $\mathbf{Z}$ mod p is chosen randomly, and random $1 < c < p - 1$.

Bob (as in ElGamal-style PK) picks random bit i and random $1 < x < p - 1$ and computes

$$\begin{array}{rcl} b_i & = & g^x \bmod p \\ b_{1-i} & = & c \cdot g^{-x} \bmod p \end{array}$$

Whichever of $0, 1$ bit i is, bit $1 - i$ is the opposite. Bob uses (b_0, b_1) as his public encryption key and keeps (i, x) secret.

Alice checks $b_0 b_1 = c \bmod p$.

Alice randomly picks y_0, y_1 in interval $[2, p - 2]$ and computes

$$a_0 = g^{y_0} \qquad t_0 = b_0^{y_0} \qquad m_0 = s_0 \oplus t_0$$

$$a_1 = g^{y_1} \qquad t_0 = b_1^{y_1} \qquad m_1 = s_1 \oplus t_1$$

She sends a_0, a_1, m_0, m_1 to Bob.

With the *secret* random bit i Bob acquires secret s_i by computing (mod p)

$$a_i^x = (g^{y_i})^x = (g^x)^{y_i} = b_i^{y_i} = t_i$$

and then

$$s_i = m_i \oplus t_i$$

Remark: Bob cannot compute the log of c base b mod p, so cannot compute the logs of *both* b_0 and b_1. The random bit i prevents public knowledge of which of b_0, b_1 Bob knows the discrete logarithm. Bob cannot acquire both s_0 and s_1, since he cannot (easily) compute the discrete log of c.

Zero-knowledge proofs

Peter (the *prover*) can convince Vera (the *verifier*) he knows the factorization of a large integer n, the product of two large primes p, q, *without* telling Vera the factorization.

Suppose $n = pq$ with large primes p, q both 3 mod 4.

Vera chooses random x and sends $x^4 \,\%\, n$ to Peter.

Peter computes the principal square root $y_1 = (x^4)^{(p+1)/2}$ of x^4 mod p and principal square root $y_2 = (x^4)^{(q+1)/2}$ of x^4 mod q, and uses Sun Ze (and Euclid) to compute y so that $y = y_1 \bmod p$ and $y = y_2 \bmod q$. Peter sends this to Vera.

Since Vera already can compute x^2, Peter has given no new information to Vera.

Vera *should* be convinced that there is no *other* way for Peter to have found this square root than by knowing the factors p, q, because in any case being able to take square roots modulo n gives a probabilistic algorithm for factoring n (when n is of the special form $n = pq$ with distinct primes p, q), as we have seen.

By the same trick, Vera can *cheat*, and learn p and q, by giving Peter not x^4 mod n but x^2 mod n... But perhaps Vera does not *want* to know p and q.

If Vera is *supervised* by a trusted third party then that third party could affirm that she did not cheat.

e-money, identity

Non-repudiation matters when signing contracts or authorizing actions electronically.

There is the issue of **identity**: Who are you? (in terms of the internet) and how do you prove it?

And can you maintain different identities for different purposes? Will that become impossible? Illegal? (In the face of anti-terrorism actions?)

Data privacy concerns medical records, credit records, records of actions taken on the internet (often *cookies*).

How can one execute a transaction on the internet without exposing more information than desired?

From the viewpoint of merchants, some information must be divulged in order to establish sufficient *trust* to justify shipping a product.

Ideally, the buyer should be able to give just enough information, no more, to the seller, in order to convince the seller to complete the transaction.

Symmetrically, the seller also may need to establish trust.

A basic technical issue in electronic transactions is *failure mode*: what happens if an electronic transaction is interrupted before completion? Is your money withdrawn from your bank, or not? Perhaps it is withdrawn from your account, but doesn't make it to you?

Protocols and software and hardware ensuring *atomicity* of transactions are important to minimize these problems.

Much as *database* designers *rollback* an incomplete transaction rather than abandoning a transaction part way through.

A true electronic abstraction of ‘money’ is still in the future.

A drawback to use of conventional credit cards, as opposed to paper money, is that the credit card’s value exists only insofar as it is *connected to your identity.*

But then your transactions are traceable.

By contrast, paper money has value in itself, and no connection to your identity, so cannot yield information about other transactions.

Current electronic banking has the same problem as credit cards: *someone* knows a lot about your financial transactions, where and when they are made, and your general spending patterns. This information can be marketed!

Ideally, *e-money* would

- have a value independent of identity
- be *divisible*
- be *transferable*
- *not* be *reusable* (can't spend the same dollar twice)
- would *not* depend on a *central authority*
- allow transactions *offline*.

We are not close to achieving these goals.