# Computer Science Tripos part 1b — Security

Lecture 4 — Public Key Basics

Ross J. Anderson

## 1 Introduction

In the last few lectures, we started to appreciate that key management is often the hardest problem in building cryptographic systems. One of the tools which we can use to tackle this problem is public key cryptography.

Public key cryptology was invented in GCHQ in the early 1970s, and rediscovered in 1975 by Diffie and Hellman at Stanford. The idea behind public key encryption is that one can find a pair of keys — an encryption key and a decryption key — with the property that the former is derived from the latter using a function that is hard to invert. The effect is that you can publish your encryption key (public key) and keep your decryption key private. Then anyone can look up your public key (e.g. on your web page) and send you confidential messages. The original GCHQ technical reports are appended.

Diffie and Hellman added the concept of a digital signature. Here one acts on cleartext data using a private key (the signature key) and anyone can verify the signature using the corresponding public key (the verification key).

## 2 Number Theory

The public key algorithms used today are overwhelmingly based on number theory, so we must assimilate a few results from this branch of mathematics. Further details may be found in any number theory textbook.

### 2.1 Primes and factoristaion

The natural numbers $\mathbf{N}$ are the positive whole numbers $\{1, 2, 3, 4, 5, ... \}$, and the prime numbers are those of them which have no proper divisors. By definition, 1 is not prime; so the primes are $\{2, 3, 5, 7, 11, ...\}$.

**The fundamental theorem of arithmetic** states that each natural number greater than 1 factors into prime numbers in a way which is unique up to the order of the factors. This follows from the fact that if a prime $p$ divides $ab$, then it divides either $a$ or $b$. If $n$ were the smallest number with two distinct prime factorisations, $n = p_1^{a_1} p_2^{a_2} ... p_k^{a_k} = q_1^{b_1} q_2^{b_2} ... q_l^{b_l}$, we would have $p_1$ dividing one of the $q_i$ and thus equal to that $q_i$, and could divide out by $p_1$ on both sides and get a smaller number with two different factorisations, which contradicts our assumption.

**The number of primes:** If there were only $k$ primes 2,3,5, ..., $p_k$, then the number $q = 2.3.5...p_k + 1$ would be either a prime, or divisible by a prime greater than $p_k$, which is a contradiction. Thus there are infinitely many primes.

The above two theorems are due to Euclid. Another result known to the ancients is the sieve of Eratosthenes; in order to construct a table of all the primes up to some limit $L$, we write down the numbers

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, ..., $L$

We now mark 2 as the first prime number, and strike out all its multiples, leaving:

**2**, 3, 5, 7, 9, 11, 13, 15, 17, ..., $L$

Next, we write down 3 as the second prime number, and strike out all its multiples, leaving:

**2**, **3**, 5, 7, 11, 17, ..., $L$

Continuing in this way, we remove the multiples of all the primes up to $\sqrt{L}$, and end up with a list of all the prime numbers less than $L$.

**The density of primes:** We write $\pi(x)$ for the number of prime numbers up to and including $x$. Thus, for example, $\pi(10) = 4$ (2,3,5,7). It is known that

$$\pi(x) \sim \frac{x}{\log x}$$

although the proof is quite hard (see 'The Theory of Numbers', by Hardy and Wright, for details). The interest to cryptologists is that this tells us the probability of a random $M$-bit number's being prime, namely about $\log_2(e)/M$. This enables us to estimate the running time of algorithms which search for large prime numbers by draw-and-test.

## 2.2 Modular arithmetic

We write $a \equiv b \pmod{n}$ if $a - b$ is divisible by $n$. This is an equivalence relation, and we can do addition, subtraction and multiplication modulo any number $n$ without difficulty. Division, however, can fail if $n$ is composite; we cannot divide by any factor of $n$.

**Fermat's (little) theorem:** If we take the set $\{1, 2, ..., p-1\}$ and multiply each of them modulo $p$ by some number $a$, then all this does is to permute them. For example,

$$\begin{array}{lll} 3.1 & \equiv 3 & \pmod 7 \\ 3.2 & \equiv 6 & \pmod 7 \\ 3.3 & \equiv 2 & \pmod 7 \\ 3.4 & \equiv 5 & \pmod 7 \\ 3.5 & \equiv 1 & \pmod 7 \\ 3.6 & \equiv 4 & \pmod 7 \end{array}$$

Thus $\{a.1, a.2, ..., a.(p-1)\} = a^{p-1}.\{1, 2, ..., p-1\}$ and by division we get Fermat's theorem: if $p$ is a prime, and $p$ does not divide $a$, then

$$a^{p-1} \equiv 1 \pmod p \tag{1}$$

**The order of a number:** If $d$ is the smallest number such that $a^d \equiv 1 \pmod n$, we say that the order of $a$ mod $n$ is $d$, $\mathrm{ord}_n(a) = d$. So Euler's theorem can be written as

$$\mathrm{ord}_n(a) \mid \phi(n)$$

Here, we use the symbol | for 'divides'.

**Discrete Logarithms:** A primitive root modulo $p$ is a number whose powers generate all the nonzero numbers mod $p$; for example, when working modulo 7 we find that

$$\begin{array}{llll} 5^1 & & = 5 & \pmod 7 \\ 5^2 = & 25 & \equiv 4 & \pmod 7 \\ 5^3 \equiv & 4.5 & \equiv 6 & \pmod 7 \\ 5^4 \equiv & 6.5 & \equiv 2 & \pmod 7 \\ 5^5 \equiv & 2.5 & \equiv 3 & \pmod 7 \\ 5^6 \equiv & 3.5 & \equiv 1 & \pmod 7 \end{array}$$

Thus 5 is a primitive root modulo 7. This means that given any $y$, we can always solve the equation $y = 5^x \pmod 7$; $x$ is then called the discrete logarithm of $y$ modulo 7.

## 2.3 Hard problems in number theory

Two number theoretic problems in particular have been very extensively studied over the years and are believed to be hard for large numbers. The first of these is calculating discrete logs modulo a large prime $p$. This is only feasible if $p-1$ is smooth (i.e., it consists only of small factors); but the theory of prime distribution assures us that primes with this property are rare, and if we choose $p$ to be a random prime number of about 1000 bits in length, then barring new developments in mathematics we can expect that no-one will be able to calculate discrete logarithms mod $p$, even given likely increases in computer power over the next 20 years.

The second hard problem is factoring. We can easily multiply together two large prime numbers $p$ and $q$ to get a composite number $n = pq$; however, it is very hard to factor a large number $n$ into its component primes. The best currently known algorithms for a number $n$ of the form $pq$ might work for $n$ up to about 500 bits in length — the largest number of this form to have been actually factored was 425 bits long, and this job took took thousands of MIPS years of distributed processing. 512 bit numbers could be factored now using the same software, at a cost of about half a million MIPS-years of effort and about 1Tb of disc storage. However the opinion among mathematicians is that if we take $p$ and $q$ to be random primes of about 500 bits, so that their product $n = pq$ is about 1000 bits long, we can reasonably expect that (again barring new mathematics) no-one will be able to factor $n$ within twenty years.

# 3 Public Key Cryptography

In a famous paper called "New Directions in Cryptography", Whitfield Diffie and Martin Hellman introduced the public to the concept of public key cryptography (this paper is appended). Apart from the concept of public key encryption (which was already known to the intelligence community) and digital signature (which wasn't), the main technical contribution was the Diffie-Hellman protocol.

## 3.1 Diffie Hellman

In order to get an insight into how we might build a cryptosystem that does not need the users to start off with a shared secret key, consider the following scenario.

Imagine that Alice wants to send a secret to Bob, and the only communications channel available is an untrustworthy courier. Alice could take the message, put it in a box, padlock it, and get the courier to take it to Bob. Bob could then put his own padlock on it too, and send it back to Alice. She could then remove her padlock, and send it back to Bob, who at last could open it.

Exactly the same can be done using the discrete logarithm problem to hide the data instead of a padlock. Alice chooses as her message a primitive root $g$ modulo a large prime $p$. Alice chooses a random number $R_A$, calculates $g^{R_A}$ modulo $p$ and sends it, together with $p$, to Bob. Bob likewise chooses a random number $R_B$ and forms $g^{R_A R_B}$ modulo $p$, which he passes back to Alice. Alice can now remove her exponentiation: using Fermat's theorem, she calculates $g^{R_B} = (g^{R_A R_B})^{p-R_A}$ modulo $p$ and sends it to Bob. Bob can now remove his exponentiation too, and so finally gets hold of $g$. The security of such schemes depends on the difficulty of the discrete logarithm problem.

In practice, it is inconvenient to encode a message to be a primitive root, and there is a simpler means of achieving the same thing. The Diffie-Hellman scheme uses $g^{R_A R_B}$ modulo $p$ as the key to a conventional encryption system, which might be based on DES.

Here, the prime $p$ and generator $g$ are typically common to all users. As before, Alice chooses a random number $R_A$, calculates $g^{R_A}$ and sends it to Bob; Bob does the same, choosing a random number $R_B$ and sending $g^{R_B}$ to Alice; they then both form $g^{R_A R_B}$, which they use as a session key. In our protocol notation,

| | | | |
|---|---|---|---|
| Message 1 | $A \rightarrow B:$ | $g^{R_A}$ | $\pmod p$ |
| Message 2 | $B \rightarrow A:$ | $g^{R_B}$ | $\pmod p$ |
| Message 3 | $A \rightarrow B:$ | $\{M\}_{g^{R_A R_B}}$ | |

This protocol has a large problem, and a small one. The large problem is that although Alice and Bob end up with a session key, neither of them has any idea who they share it with.

Suppose that in our padlock protocol Charlie had just bribed the courier to bring the box to him instead, and placed his own padlock on it next to Alices's. After she has removed her padlock, the courier brings the box back to him, and he can now open it.

In the Diffie Hellman protocol, the same idea leads to the middleperson attack. Charlie intercepts Alice's message to Bob and replies to it; at the same time, he initiates a key exchange with Bob, pretending to be Alice. He ends up with a key $g^{R_A R_C}$ which he shares with Alice, and another key $g^{R_B R_C}$ which he shares with Bob. So long as he continues to sit in the middle of the network and translate the messages between them, they may have a hard time detecting that their communications are compromised.

In one secure telephone product, the two principals would read out an eight digit hash of the key they had generated and check that they had the same value before starting to discuss classified matters. However, this is not enough, because of an additional small problem — the smooth subgroup attack.

## 3.2 Choice of $g$

The multiplicative group modulo $p$ will have order $p-1$, and so it will have a subgroup of every order dividing $p-1$. Suppose for the sake of simplicity that $p-1 = qw$ where $q$ is a large prime and $w$ is smooth, i.e. composed of small factors (since $p-1$ is even it will always have a smooth factor, namely 2). An attacker will be able to compute discrete logs in the group of order $w$.

So if Charlie is able to modify the traffic between Alice and Bob, he can intercept Alice's message $g^{R_A}$ and replace it with $g^{R_A q}$. Similarly, he will replace Bob's message $g^{R_B}$ with $g^{R_B q}$ and the two principals will end up both computing $g^{R_A R_B q}$ as their session key. However, Charlie can calculate $R_A \pmod w$ using the techniques described in the above paragraph, and so calculate $g^{R_A R_B q}$. There are two basic ways to fix this problem — either operate in a prime order subgroup of the integers modulo $p$, or authenticate not the shared key, but the components $g^{R_A}$ and $g^{R_B}$ that the two principals thought they had exchanged. In general, it is a good idea to operate in a group of prime order as smooth subgroups can give rise to various other problems (see the exercises).

## 3.3 ElGamal encryption

Variants of Diffie Hellman are widely used, because a key can be set up easily and also because once a key has been used and destroyed by both parties, it can never be recovered — not even if all the key distribution centres are compromised. This property is called 'forward secrecy'.

Diffie and Hellman proposed that their protocol be used to provide a public key encryption scheme by the simple means of having each user choose a private key $X_A$ say and placing $Y_A = g^{X_A}$ in a public directory, like a phone book. (From here on, we will take the phrase 'modulo $p$ as assumed.) Then if Bob wants to speak to Alice, he can look up her public key $Y_A$ and raise it to the power of his secret key $X_B$ getting the shared communication key $Y_A^{X_B} = g^{X_A X_B}$. Alice can also compute this by raising Bob's public key $Y_B$ to the power of her secret key $X_A$.

A variant, due to ElGamal, uses an extra key for each message. Let Alice's private key be $X$, and her public key $Y = g^X$. Then in order to encrypt a message $M$ to send to her, the idea is to send $BM$, hint$(B)$ where $B$ is a blinding factor. A suitable blinding factor is $y^k$, where $k$ is a randomly chosen session key. In detail, the encrypted message is

$$g^k, Y^k M$$

Alice can decrypt this by raising the hint $g^k$ to the power $X$ (her secret key), getting $Y^k$; she can then divide out the second term to recover $M$.

## 3.4 One-time Signatures

We do not need number theory to construct digital signatures; it is sufficient to have a strong hash function, as discussed in the previous lecture. The user chooses two sets of random numbers $R_i$, $S_i$ and publishes as his public verification key the vectors $h(R_i)$, $h(S_i)$ where $h$ is a suitable one-way function. (This function must be 'collision-free' in the sense that it is not computationally feasible to find two different inputs $R_1 \neq R_2$ such that $h(R_1) = h(R_2)$.)

The signature on a message $M$ whose bits are $M_i$ consists of the values $R_i$ for those bits $M_i$ which are 1, and $S_i$ for those bits $M_i$ which are 0.

This appears to require a large number of key bits for each message bit which we wish to sign. However, various optimisations are possible, which include:

- we hash each message before signing, so that instead of a string $M$ of arbitrary length we only sign a hash $h(M)$ of say 128 bits;
- rather than having two vectors $R_i$, $S_i$ we have just one, and sign only those bits whose value is 1. In this case, we need to include a further short signature of the total number of nonzero bits (why is this?) giving a total overall signature length, for an $n$ bit hash, of $n \log n$ hashes rather than $2n$;
- rather than publishing the whole vector $\{h(R_i)\}$ we publish its hash, so the public key at least can be short. Of course we then have to send the vector $\{h(R_i)\}$ along with the signature so it can be verified.

These 'one-time signatures' as they are called were among the first signature schemes to be discovered. They are used, for example, in an electronic publishing system developed at Addenbrooke's in which each version of a publisher's catalogue contains a small number of one-time public keys by which future versions of the catalogue can be authenticated. The advantages of one-time signatures are principally that they are simple and fast to compute; the big disadvantage is that each public/private keypair can only be used once.

### 3.5 ElGamal signatures

Number theory can be used to construct digital signature schemes in which each keypair can be used a large number of times. An example is the ElGamal signature scheme. As with ElGamal encryption, we use a message key $k$ and send a hint $g^k$; we will again assume that the user's private signing key is $X$ and her public signature verification key is $Y = g^X$. As with one-time signatures, we will hash the message $M$ to $h(M)$ for performance reasons. The signature will be the two numbers $r$ and $s$ where $r = g^k$ and $s$ is given by a linear relation. The signature is computed by solving, modulo $p-1$,

$$rX + sk = h(M)$$

in other words,

$$s = (h(M) - rX)/k$$

In order to verify this signature, the generator $g$ is raised to the power of both sides of the equation:

$$g^{rX} g^{sk} = g^{h(M)}$$

or

$$y^r r^s = g^{h(M)}$$

As with Diffie-Hellman, there are technical reasons why we prefer to choose $g$ with care (it should be the generator of a prime order subgroup). This detail is explored in the exercises.

### 3.6 The Digital Signature Standard

The US digital signature standard, which you may see referred to in the literature as DSA or DSS, is a variant on the ElGamal signature scheme and is used widely in US military systems. It assumes a prime $p$ of 512–1024 bits, a prime $q$ of 160 bits dividing $(p-1)$, an element $g$ of order $q$ in the integers modulo $p$, a secret signing key $x$ and a public verification key $y = g^x$. The signature on a message $M$ is $(r, s)$ where

$$r = (g^k \pmod p) \pmod q$$

$$s = (h(M) - xr)/k \pmod q$$

### 3.7 RSA

Many commercial public-key-encryption and digital signature systems use RSA, an algorithm based on the difficulty of factoring rather than on the difficulty of the discrete logarithm problem. We do not have time to explore this as well but will discuss it in part 2. The curious can refer to Schneier for details; a more mathematical exposition can be found in Koblitz.

## 4 Public Key Protocols

Now that we can do public key encryption and digital signature, we need some mechanism to bind users to keys. One approach is to have a directory of the keys of a system's authorised users; another is for the system administrator to sign the public encryption and/or signature verification keys of the users. In this way, only the administrator's public signature verification key needs to be known to all principals in a trustworthy manner.

One of the first proposed public key protocols was due to Denning and Sacco, who in 1982 proposed that each user $A$ should have a certificate of the form

$$CA = \{T_S, L, A, K_A\}_{K_S^{-1}} \tag{2}$$

where, as with Kerberos, $T_S$ is the certificate's starting date and time, and $L$ is the length of time for which it is valid. They further proposed that users set up a shared DES key $K_{AB}$ as follows:

$$\begin{array}{ll} A \rightarrow S: & A, B \\ S \rightarrow A: & CA, CB \\ A \rightarrow B: & CA, CB, \{\{T_A, K_{AB}\}_{K_A^{-1}}\}_{K_B} \end{array}$$

In 1994, Abadi and Needham pointed out that this protocol is fatally flawed. Bob, on receiving this message, can masquerade as Alice for as long as Alice's timestamp $T_A$ remains valid!

To see how, suppose that Bob wants to masquerade as Alice to Charlie. He goes to Sam and gets a fresh certificate $CC$ for Charlie, and then strips off the outer encryption $\{...\}_{K_B}$ from message 3 in the above protocol. He now re-encrypts the signed key $\{T_A, K_{AB}\}_{K_A^{-1}}$ with Charlie's public key — which he gets from $CC$ — and makes up a bogus message 3:

$$B \rightarrow C: \quad CA, CC, \{\{T_A, K_{AB}\}_{K_A^{-1}}\}_{K_C}$$

The design of public key protocols, as with conventional encryption protocols, is much harder than it looks.

# 5 Problems

1. Show that if the generator $g = 2$ is chosen, then ElGamal signatures can be forged on any message.

2. (Hard.) Show that if the generator $g$ is a primitive root, then every ElGamal signature verification key has a *compatible weak key*, that is, a key which will verify as valid all signatures created using the corresponding genuine signature key, but which will also verify signatures generated with another signature key that can be computed by an attacker.

   What sort of applications of digital signatures might be threatened by such a vulnerability in the signature scheme?

3. Write down the equation(s) used to verify a DSS signature.

   The standard also specifies that $r$ must lie strictly between 0 and $q$. What might go wrong if an application developer failed to check this?

   Show that if the prime $q$ is chosen as the difference of two message hashes, i.e. $q = h(M_1) - h(M_2)$, then $M_1$ and $M_2$ will have the same DSS signature. Is this a vulnerability?

4. The Schnorr signature is like DSA an ElGamal variant; like DSA it uses a generator $g$ of order $q$ where $q$ is a 160 bit prime divisor of $p-1$. The message key $k$ is hashed with the message $M$ to give

$$r = h(M, g^k)$$

The second part of the signature, $s$, is computed as

$$s = k + rX$$

Explain how this signature is verified. Discuss the relative computational costs of DSA and Schnorr signatures for computation and verification.

5. Needham and Schroder proposed a key exchange protocol that runs as follows. Here the encryption operation $\{X\}_{KA}$ means the value $X$ encrypted under Alice's public key, and $NA$ is a random challenge generated by Alice. The object of the protocol is that, at its conclusion, the principals Alice and Bob end up being able to use the value $NA \oplus NB$ as a DES key.

   $A \longrightarrow B : \{N_A, A\}_{K_B}$
   $B \longrightarrow A : \{N_A, N_B\}_{K_A}$
   $A \longrightarrow B : \{N_B\}_{K_B}$

   What is wrong with this protocol?

6. In X.509, Alice may sign a message of the form $\{T_A, N_A, B, X, \{Y\}_{K_B}\}$ and send it to Bob. Here $T_A$ is a timestamp, $N_A$ is a serial number, and $X$ and $Y$ are user data. This is the reverse of the order used in paper correspondence (where one first signs a letter and then puts it into an envelope). Discuss the relative merits of encrypting signed data, and signing encrypted data. Which is better for normal commercial traffic? Under what circumstances might the order of encryption and signature be reversed?