TALK: PUBLIC KEY CRYPTOGRAPHY

ALAN BRACE

Contents

1. Signed Contracts

1.1. **Kerckhoff's principle.** Now a cliche, but not 40 years ago!!! - make all algorithms public because best way find holes and ensure security - plus keeps NSA (National Security Agency) on toes

1.2. **Symmetric key cryptography SKC.** Alice and Bob both have same secret key to communicate - i.e. same key used to encrypt and decrypt - symmetric! - world of *one time pads* of spy novels - in 1977 US NBS (National Bureau Standards) put out 56-bit key DES (Data Encryption Standard) - were suspicions that NSA had influenced negatively but later decided no so - in 1998 *Deep Crack* brute-force broke DES in 56 hours

1.3. **Advanced Encryption Standard AES.** Present standard for SKC is (usually) 256-bit key AES - approved as a US standard in 2001 - expected last several decades - in 2003 NSA announced allow AES to encrypt govt docs up to TOP SECRET if using 256-bit keys

1.4. **Key distribution problem.** In SKC every pair people need a separate key - n people $\frac{1}{2}n(n-1)$ keys - so lot of keys - and key distribution problem - bad-guy Oscar can intercept, steal, compromise etc

1.5. **Public key cryptography PKC.** Bob want send message x to Alice - in PKC Alice has published her public encryption key e - Bob (or anybody) uses it to encrypt a message $e(x)$ and sends to Alice - Alice then uses her private decoding key d to get $d[e(x)] = x$ - note asymmetry, if e encodes then d decodes, while if d encodes then e decodes - the pair e, d are mutually inverse

At present 3 avenues to implementation

(1) RSA (Rivest, Shamir, Adleman) by factorization of large numbers
(2) DL discrete logarithm approach
(3) ECC - Elliptic curve cryptography

Date: Wed 25 May 2016 *Email*: alan.brace@nab.com.au.

We will look at RSA in detail, but **just for the book** the second two work with various cyclic groups $\langle G, \circ \rangle$ of cardinality n and with generator α, which means that if α is repeatedly multiplied by itself

$$\alpha \circ \alpha \circ \alpha \circ ..... \circ \alpha = \alpha^m$$

the resutl runs through every element in the group - for example $\circ = \times \mod 11$ and $\alpha = 2$ in

$$G = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$
$$i = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$
$$2^i = \{2, 4, 8, 5, 10, 9, 7, 3, 6, 1\}$$

note how generated elements almost random

The **discrete logarithm problem** is given some $\beta \in G$ find m such that

$$\alpha^m = \beta \qquad \text{written} \qquad m = \log_\alpha \beta$$

For cyclic groups of large cardinality (like 10^{200}) or with a cunningly devised group operation $\circ$ (as in ECC), this can be difficult

1.6. **Man-in-the-middle problem with PKC.** Oscar fools Bob that his key is Alice's - Bob sends message - Oscar intercepts and decodes - then Oscar sends on corruption to Alice using her public key - neither Alice nor Bob realize been compromised - so must be sure of owner of public key - publish in paper, government can certify, known friend can cryptographically sign

1.7. **PKC much slower than SKC.** Keys are computationally secure if they take many years using the latest algorithms on the fastest computers to crack - NSA assumes quantum computer?? - the following table shows the key sizes needed to obtain various equivalent security levels

Symmetric AES	80	120	192	256
RSA	1024	3072	7680	15360
DL	1024	3072	7680	15360
ECC	160	256	384	512

Here is an example of a 512 bit key in RSA (in hexadecimal - base 16)

$$p = E0DFD2C2A288ACEBC705EFAB30E4447541A8C5A47A37185C5A9$$
$$CB98389CE4DE19199AA3069B404FD98C801568CB9170EB712BF$$
$$10B4955CE9C9DC8CE6855C6123$$

These sorts of numbers are of the order of

$$10^{154} \approx (\text{trillion})^{17}$$

far far more than the number of atoms in the universe, about 10^{80} - even worse, imagine a 15360 bit key - got to keep exact track of the integers involved in corresponding algorithms - so good algorithms important - even then PKC is **much slower** than SKC by several orders of magnitude - note ECC efficient - Bitcoin uses ECC with 256 bit key

1.8. **PKC in practice.** In practice PKC and SKC are both used together - Alice constructs an SKC key, then uses PKC to send it to Bob, after which they both use SKC to communicate - much faster - and the SKC key distribution problem is solved!!!

1.9. **Verifiable signatures.** PKC allows messages to be signed and verified - **this the basis for all electronic contracts, deals, trading** etc - **essential for blockchains** (and Bitcoin) - SKC can't do this - works as follows:

(1) Alice sends the contract as an open text message x to Bob
(2) Then she *signs* the contract by encoding the message with her private decoding key d and sending that to Bob, who now also has $d(x)$ as well as x
(3) Bob uses Alice's public encoding key e to decode the signed message and compare with original

$$e[d(x)] = x \qquad or \qquad e[d(x)] \neq x \quad ???$$

If identical, then the contract is authenticated - **only Alice knows d so only Alice could have sent it**
(4) For a **jointly signed contract**, Bob sends his signature to Alice in a similar fashion

Note that using SKC Bob **cannot prove** a message came from Alice - he has the same symmetric key as Alice and could have **forged** the signature

1.10. **Signatures in practice.** Messages may long and confidential - so in practice something like following used

(1) Alice encrypts the message x using their established SKC key S and sends to Bob, who gets $S(x)$ and recovers x by $S^{-1}\{S(x)\} = x$
(2) Alice **hashes** h (a hash is a public function that takes a message and produces a unique short totally scrambled digest of a message - more later) the encrypted message $h[S(x)]$, encrypts that with her private decoding key d and sends to Bob who now also has $d\{h[S(x)]\}$
(3) Bob decodes $d\{h[S(x)]\}$ with Alice's public key e, hashes $S(x)$ and compares - is

$$e(d\{h[S(x)]\}) = h[S(x)] \quad ???$$

Message, contract, deal etc confidentially received and authenticated

1.11. **Fading ink problem.** Ink fades for a paper signature - and after many years the PKC implementation behind a signature may become vulnerable to attack because of better algorithms or faster computers

2. The RSA implementation of PKC

2.1. **To begin RSA.** Start by choosing two (large like at least 512 bits long) prime numbers p and q and then work in the related *integer rings*

$$\mathbb{Z}_n \qquad n = p \times q \qquad \mathbb{Z}_n = \{0, 1, 2, ..., n-1\} \qquad arithmetic \qquad \bmod n$$
$$\mathbb{Z}_m \qquad m = (p-1) \times (q-1) \qquad \mathbb{Z}_m = \{0, 1, 2, ..., m-1\} \qquad arithmetic \qquad \bmod m$$

Modular arithmetic is easy "mod(sum/product)=sum/product(mod)"

$$a \mod n = c, \quad b \mod n = d \implies a = s \times n + c, \quad a = t \times n + d$$
$$(a+b) \mod n = [(s+t) \times n + c + d] \mod n = [c+d] \mod n$$
$$(a \times b) \mod n = [(s.t.n + s.d + t.c) \times n + c \times d] \mod n = [c \times d] \mod n$$
$$e.g.\ 103 \times 277 \mod 20 = 3 \times 17 \mod 20 = 11$$

2.2. **Simple example.** In our simple example we choose $p = 3$, $q = 11$ so $n = 33$ and $m = 2 \times 10 = 20$ - thus in $\mathbb{Z}_{20}$ arithmetic is $\mod 20$ so

$$a + b = c \mod 20, \quad c \in \mathbb{Z}_{20} \qquad a \times b = d \mod 20, \quad d \in \mathbb{Z}_{20}$$

$$4 + 6 = 10, \quad 6 + 19 = 5, \quad 9 + 0 = 9, \quad 9 + 11 = 0, \quad 5 + 15 = 0,$$

$$4 \times 6 = 4, \quad 6 \times 19 = 14, \quad 9 \times 0 = 0, \quad 9 \times 1 = 9, \quad 11 \times 11 = 1, \quad 17 \times 13 = 1$$

All operations as for numbers **except for multiplicative inverses** e.g. can't find x such that $8 \times x = 1 \mod 20$

2.3. **Multiplicative inverses.** Number x has **multiplicative inverse** in $\mathbb{Z}_n$ if and only if the greatest common divisor of x and n is 1

$$\text{i.e. iff} \qquad \gcd(x, n) = 1$$

For example, $\gcd(13, 20) = 1$ got inverse 17, but $\gcd(8, 20) = 4$ so can't find 8^{-1}

2.4. **Euclidean algorithms.**

(1) First algorithm says if $q < p$ then

$$\gcd(p, q) = \gcd(p - q, q)$$

Easy to see because $\gcd(p, q)$ means $p = a \times g$ and $q = b \times g$ so $p - q = (b - a) \times g$ - note extension if $r \times q < p$ then repeating algorithm gives

$$\gcd(p, q) = \gcd(p - r \times q, q)$$

So can get numbers down fast - good algorithm to test $\gcd(p, q) = 1$

(2) Second says we can always find integers s and t such that

$$\gcd(p, q) = s \times p + t \times q$$

Easy to see, for example

$$\gcd(13, 20) = \gcd(7, 13) = \gcd(7, 6) = \gcd(7, 1) = 1$$

$$1 = 7 - 6 = 7 - (13 - 7) = 2 \times (20 - 13) - 13 = 2 \times 20 - 3 \times 13$$

$$\implies \quad (-3 \times 13) \mod 20 = 1 \quad \implies \quad -3 \mod 20 = 17 = 13^{-1}$$

Good algorithm for finding inverses

2.5. **Euler's Phi function $\Phi(n)$.** The number of integers in $\mathbb{Z}_n$ relatively prime to

$$n = p_1^{e_1} \times p_2^{e_2} \ldots \times p_k^{e_k} \quad \text{is} \quad \Phi(n) = \prod_{i=1}^{k} (p_i^{e_i} - p_i^{e_{i-1}}) \qquad \blacksquare$$

$$\implies \quad \Phi(p) = (p - 1) \qquad \Phi(q) = (q - 1) \qquad \Phi(pq) = (p - 1) \times (q - 1)$$

So $n = 33 = 3 \times 11 \implies \Phi(33) = 2 \times 10 = 20$ and $\mathbb{Z}_{33}$ must have 20 inverses
And $m = 20 = 2^2 \times 5 \implies \Phi(20) = (2^2 - 2^1) \times 4 = 8$ so $\mathbb{Z}_{20}$ has 8 elements with inverses, namely the elements

$$\{1, 3, 7, 9, 11, 13, 17, 19\}$$

2.6. **Euler's Theorem.** This theorem says that if x and n are integers with $\gcd(x,n)=1$ then

$$x^{\Phi(n)} \mod n = 1 \implies x \times x^{\Phi(n)-1} \mod n = 1 \implies x^{-1} = x^{\Phi(n)-1}$$

giving $x^{\Phi(n)-1}$ as the inverse of x in $\mathbb{Z}_n$

For example, in $\mathbb{Z}_{20}$ $\Phi(20)=8$, so the inverse of 13 in $\mathbb{Z}_{20}$ will be $13^7 \mod 20 = 17$ which can be computed like

$$13^2 \mod 20 = 9, \qquad 13^4 \mod 20 = 1, \qquad \implies$$

$$13^7 \mod 20 = \left(13^4 \mod 20 \times 13^2 \mod 20 \times 13^1\right) \mod 20 = 117 \mod 20 = 17$$

Observe $7 \equiv 111$ in binary, giving good exponentiation algorithm

2.7. **Example RSA with keys in $\mathbb{Z}_{20}$ messages in $\mathbb{Z}_{33}$.** Chose encoding key $e = 13 \in \mathbb{Z}_{20}$ with decoding key d such that

$$e \times d \mod 20 = 1 \qquad i.e. \qquad d = e^{-1} = 17 \in \mathbb{Z}_{20}$$

Let the message be $x = 26 \in \mathbb{Z}_{33}$ (representing say 'z') which encodes and decodes like

$$y = 26^{13} \mod 33 = 20 \qquad \rightleftarrows \qquad x = 20^{17} \mod 33 = 26$$

Encode $13 \equiv 1101$ giving (her $x \times y \equiv$z means $x \times y \mod 33 = z$)

$$26^2 \equiv 16, \quad 26^4 \equiv 25, \quad 26^8 \equiv 31, \quad 26^{13} \equiv (31 \times 25 \times 26) \equiv 20$$

Decode $17 \equiv 10001$giving

$$20^2 \equiv 4, \quad 20^4 \equiv 16, \quad 20^8 \equiv 25, \quad 20^{16} \equiv 31, \quad 20^{17} \equiv (20 \times 31) \equiv 26$$

2.8. **RSA in general.** Keys $e, d \in \mathbb{Z}_m$, messages $x, y \in \mathbb{Z}_n$**,** exponentiation is $\mod n$

(1) The RSA **encoding** key e is chosen from among the elements in $\mathbb{Z}_m$ that are relatively prime to m i.e. $\gcd(x,m)=1$
(2) Its **decoding** key d is the inverse of e in $\mathbb{Z}_m$ i.e. $e \times d \mod m = 1$ or $d = e^{-1}$
(3) A **message** is an element $x \in \mathbb{Z}_n$ encoded to $y \in \mathbb{Z}_n$ and then decoded back to x like

$$y = x^e \mod n \qquad \rightleftarrows \qquad x = y^d \mod n$$

2.9. **Why RSA works.** By construction we know there is an integer t such that

$$(e \times d) \mod \Phi(n) = 1 \implies e \times d = 1 + t \times \Phi(n)$$

and we want to show that we can get back

$$x = (x^e)^d \mod n = x^{e \times d} \mod n = x \times x^{t \times \Phi(n)} \mod n$$

So if we can prove that

$$x \times x^{\Phi(n)} \mod n = x$$

for all $x \in \mathbb{Z}_n$ then we are done because

$$x \times x^{2\Phi(n)} \mod n = \left[\left(x \times x^{\Phi(n)} \mod n\right) \times x^{\Phi(n)} \mod n\right] \mod n = x \times x^{\Phi(n)} \mod n = x$$

and so on t times

Case 1: When $\gcd(x,n)=1$ the result follows because Euler's Theorem says

$$x^{\Phi(n)} \mod n = 1 \implies x \times x^{\Phi(n)} \mod n = x$$

Case 2: When $\gcd(x,n) \neq 1$ recalling that $n = p \times q$ is the product of two primes p and q, we must have either

$$x = r \times p \qquad or \qquad x = s \times q$$

where r and s are integers such that $r < q$ and $s < p$ (e.g. $\gcd(24,33) \neq 1$ gives $24 = 8 \times 3$) Suppose $x = r \times p$ then $\gcd(x,q) = 1$ and Euler's Theorem gives

$$\begin{aligned}
x^{\Phi(q)} \mod q &= x^{q-1} \mod q = 1 \qquad \Longrightarrow \\
\left(x^{\Phi(n)}\right)^t \mod q &= \left(x^{q-1}\right)^{t\times(p-1)} \mod q = 1 \qquad \Longrightarrow \\
\left(x^{\Phi(n)}\right)^t &= 1 + u \times q \qquad u \quad \text{an integer} \qquad \Longrightarrow \\
x \times \left(x^{\Phi(n)}\right)^t &= x + x \times u \times q = x + r \times p \times u \times q \\
&= x + (r \times p) \times n \qquad \Longrightarrow \\
x \times x^{\Phi(n)} \mod n &= x
\end{aligned}$$

3. Implementation and problems

The math in the previous section very pretty - but how secure is RSA - and with numbers like $p, q = o\left(10^{154}\right)$ occurring, how to implement???

3.1. Security.

Available public information is the encoding key e and the modulus number n - to get the decoding key d Oscar needs $\Phi(n) = (p-1) \times (q-1)$ because $d = e^{-1} \mod \Phi(n)$ - so security of RSA rests on Oscar not being able to factorize $n = p \times q$ in a reasonable time - hence the better the latest factorization algorithm and the faster the latest computer, the longer the RSA key must be

Also RSA is **malleable** - i.e. Alice encodes message x as $y = x^e$, Oscar intercepts and replaces y with $s^e \times y$ where s is some integer - Bob then decodes

$$(s^e \times y)^d = s^{ed} \times x^{ed} = s \times x \mod n$$

scrambling the original x - Oscar can't decode, but can cause trouble!! - tackle by having a specific format for x, like that in Public Key Cryptography Standard #1 (PKCS #1).

3.2. Finding p and q.

The **prime number theorem** says that for large enough p, the probability that a random integer not greater than p is prime is very close to $1/\ln(p)$ - ruling out even numbers the probability that a 512 bit number p is prime is therefore

$$\text{Prob } p \text{ prime} = \frac{2}{\ln\left(2^{512}\right)} = \frac{2}{512\ln(2)} \approx \frac{1}{177}$$

so picking 512 bit numbers at random roughly get a prime every 200 times - and there will be at least 10^{151} primes to choose from - that doable so long as got a fast way to check if a number is prime

The **Miller–Rabin (MR) primality test** takes some random number $2 \leq a \leq p-2$, does a computation with a and p, and comes to one of two possible conclusions

(1) p is **definitely** a composite number i.e. not a prime
(2) p is **probably** a prime

So keep running the MR test on a candidate p with different a and if the answer is "probably a prime" about half-a-dozen times, then the probability that p is composite i.e.not prime, is something like 10^{-80}

3.3. **Finding e and d.** Having picked a couple of 512 bit primes p and q

(1) compute the Euler Phi function $\Phi(n) = (p-1)(q-1)$
(2) select the public encoding key $e \in \{1, 2, ..., \Phi(n) - 1\}$ such that $\gcd(e, \Phi(n)) = 1$
(3) compute the private key $d = e^{-1} \mod \Phi(n)$ as the inverse of e

Candidates for e are picked at random and tested with the first Euclidean algorithm so see if $\gcd(e, \Phi(n)) = 1$ - if positive, the second Euclidean algorithm gives the inverse decoding key d - both these algorithms are fast and efficient

3.4. **Encoding and decoding messages.** Message x is encoded and decoded by exponentiation like

$$y = x^e \mod n \qquad \rightleftarrows \qquad x = y^d \mod n$$

Recalling keys $e, d \in \mathbb{Z}_m$ and messages $x, y \in \mathbb{Z}_n$, that means we have to efficiently compute exponentiation like

$$1024 \text{ bit } y = (1024 \text{ bit } x)^{512 \text{ bit } e} \qquad 1024 \text{ bit } x = (1024 \text{ bit } y)^{512 \text{ bit } d}$$

The algorithm outlined in §2.7 is efficient enough, and a version of the **Chinese Remainder** theorem is faster

References

[1] Lidl R, Niederreiter H (1994) Introduction to Finite Fields and Their Applications, *CUP*
[2] Paar C, Pelzl J (2010) Understanding Cryptography, Springer ISBN 978-3-642-04100-6