Chapter 5: Split Keys

Panos Louridas

Athens University of Economics and Business

Real World Algorithms
A Beginners Guide
The MIT Press

Outline

The Super Trustworthy Boxes (STB) Company

- The STB company makes safe boxes with a twist.
- Instead of one key, each box has two.
- If we lock a box with one key, we can unlock it only using the other key.

Advantages of STB Boxes (1)

1. Alice wants to send something to Bob.
2. Alice asks Bob to send her an STB box along with a key.
3. Bob sends Alice an STB box and a key. He keeps the second key secret.
4. Alice puts her message in the box, locks it, and sends the box to Bob.
5. Bob gets the box, he opens it using his key, and reads Alice's message.

Advantages of STB Boxes (2)

- Eve, an eavesdropper, can intercept the box and the key sent by Bob.
- However, as she does not have Bob's secret key, she cannot open it.
- Only Bob can open the box that Alice has locked.

Advantages of STB Boxes (3)

1. Alice puts her message in an STB box.
2. Alice locks the box using one key and sends the box along with the other key to Bob.
3. Bob opens the box using the key that came along with it.
4. Only Alice could have locked the box, so Bob knows that only she can be the sender.

Advantages of STB Boxes (4)

STB boxes offer:

- Security from eavesdropping.
- Sender identification.

Outline

Symmetric and Asymmetric Cryptography

Definition

If we use the same key for encryption and decryption, we have *symmetric cryptography*.
If we use different keys for encryption and decryption, we have *asymmetric cryptography*.

Public Key Cryptography

Definition

Public key cryptography is a form of asymmetric cryptography. It works with two keys, one *public* and one *private.*

Public and Private Key

- The public key can be distributed freely.
- The secret key must always remain protected and must never be leaked.

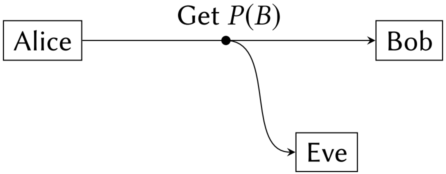
Key Pair

- The public and the private key make a pair.
- A public key can work only with the private key in its pair.
- We denote a key pair as:

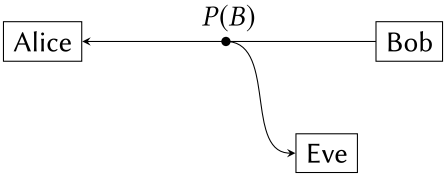
  $$A = (P(A), S(A))$$

  where $P(A)$ is the public key of pair A and $S(A)$ is the private key of pair A.

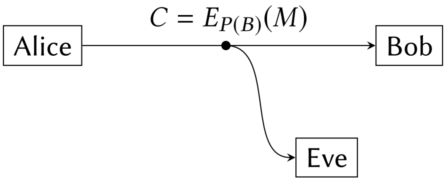
Public Key Encryption (1)



Public Key Encryption (2)



Public Key Encryption (3)



Encryption Process

1. Bob creates a key pair $B = (P(B), S(B))$.
2. Alice gets Bob's pubic key, $P(B)$. This can happen in a number of ways. For example, Bob can send it to her via e-mail, or he can place it on a public server.
3. Alice encrypts her message M using $P(B)$:

$$C = E_{P(B)}(M)$$

4. Alice sends C to Bob.
5. Bob decrypts C using his private key:

$$M = D_{S(B)}(C)$$

Digital Signature

1. Alice creates a key pair $A = (P(A), S(A))$. She gives $P(A)$ to Bob using any convenient way.
2. Alice encrypts message M using her secret key, $S(A)$:

   $$C = E_{S(A)}(M)$$

   C is Alice's *digital signature* for M.
3. Alice sends (M, C) to Bob.
4. Bob verifies C using Alice's public key, $P(A)$:

   $$M \stackrel{?}{=} D_{P(A)}(C)$$

   Bob now knows that the message was sent by the owner of $S(A)$.

Encryption and Signing (1)

1. Alice creates a key pair $A = (P(A), S(A))$. She gives $P(A)$ to Bob using any convenient way.
2. Bob creates a key pair $B = (P(B), S(B))$. He gives $P(B)$ to Alice using any convenient way.
3. Alice signs her message M using her private key, $S(A)$:

$$C_1 = E_{S(A)}(M)$$

4. Alice encrypts her message and her signature, that is (M, C_1), using Bob's public key, $P(B)$

$$C_2 = E_{P(B)}(M, C_1)$$

5. Alice sends C_2 to Bob.

Encryption and Signing (2)

6. Bob decrypts C_2 using his secret key:

$$(M, C_1) = D_{S(B)}(C_2)$$

7. Bob verifies C_1 using Alice's public key $P(A)$:

$$M \stackrel{?}{=} D_{P(A)}(C_1)$$

Bob knows that the message was sent by the owner of $S(A)$.

Outline

History

- The RSA cryptosystem was named by its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman.
- It was first published in 1977.
- A similar system was discovered in 1973 from Clifford Cocks, working for the UK's Government Communications Headquarters (GCHQ).
- Cocks's work remained secret until 1997.

RSA Encryption and Decryption

- To encrypt M we do:

$$C = E_{P(A)}(M) = M^e \bmod n$$

- To decrypt C we do:

$$M = D_{S(A)}(C) = C^d \bmod n$$

- This works thanks to the proper choice of e, d.
- The public key is $P(A) = (e, n)$.
- The private key is $S(A) = (d, n)$.

RSA Signature

- To sign M we do:
$$C = E_{S(A)}(M) = M^d \bmod n$$
- To verify the signature we do:
$$D_{P(A)}(C) = C^e \bmod n$$
- This works thanks to the proper choice of e, d.
- The public key is $P(A) = (e, n)$.
- The private key is $S(A) = (d, n)$.

RSA Key Pair Creation (1)

1. Choose two large prime numbers, p and q, such that $p \neq q$.
2. Calculate their product, $n = pq$.
3. Pick a number e relatively prime to $(p-1)(q-1) = \varphi(n)$ (we'll see what exactly is $\varphi(n)$).
4. Find a number d, $1 \leq d < \varphi(n)$, such that $(e \cdot d) \bmod \varphi(n) = 1$.
5. The public key is $P(A) = (e, n)$.
6. The private key is $S(A) = (d, n)$.
7. The key pair is $A = (P(A), S(A)) = ((e, n), (d, n))$.

RSA Key Pair Creation (2)

- The number e does not need to be large, it can even be 3.
- A good choice, having good number theoretic properties, is $e = 2^{16} + 1 = 65537$.

Euler φ Function

Definition

If n is a positive integer, the Euler phi function (or Euler totient function) is the function that counts the natural numbers that are relatively prime to n. The symbol of the function is $\varphi(n)$. If n is prime, then $\varphi(n) = n - 1$.

Modular Multiplicative Inverse

- The number d is the *multiplicative inverse modulo $\varphi(n)$ of e*, or *modular inverse* for short.
- In general, the inverse of x is the number $1/x$, or x^{-1}, such that $xx^{-1} = 1$.
- The multiplicative inverse modulo n of x is a number x^{-1}, $1 \leq x^{-1} \leq n - 1$, such that $xx^{-1} \bmod n = 1$. That is equivalent to $xx^{-1} = kn + 1$, or $1 = xx^{-1} - kn$, for some integer k.
- When x and n are relatively prime, there always exists the multiplicative inverse of x modulo n. Therefore step 4 of RSA always works, as e and $\varphi(n) = (p - 1)(q - 1)$ are relatively prime.

Modular Inverse (1)

- To calculate the multiplicative inverse modulo n of a number, we use the Extended Euclid's algorithm, which we'll see shortly.
- The Extended Euclid's algorithm calculates for two numbers a, b, three numbers d, x, y, such that:

$$d = \gcd(a, b) = xa + yb$$

- If a, b are relatively prime, $d = 1$ so:

$$1 = xa + yb$$

- Therefore $x \bmod b$ is the multiplicative inverse with modulo b of a (we write $x \bmod b$ because it is not necessary that $1 \leq x \leq b - 1$, we may have $x < 0$).

Modular Inverse (2)

- In RSA we find d, where $1 \leq d \leq \varphi(n)$, such that $1 = ed + k\varphi(n)$, where k is an integer.
- To do that we use the Extended Euclid's algorithm with input e and $\varphi(n)$.
- The algorithm returns $(1, x, y)$.
- We take $d = x \bmod \varphi(n)$.

RSA Example (1)

Suppose that Bob wants to encrypt the message $M = 314$ and send it to

1. Alice chooses $p = 17$ and $q = 23$.
2. Alice calculates $n = pq = 17 \times 23 = 391$.
3. Alice chooses $e = 3$, which is relative prime with

$$(p-1)(q-1) = (17-1)(23-1) = 16 \times 22 = 352$$

4. Alice uses the Extended Euclid's Algorithm to find d such that $1 = 3d + 352k$. She finds:

$$1 = 3 \times (-117) + 1 \times 352$$

5. Therefore, $d = (-117) \bmod 352 = 235$.

RSA Example (2)

5. The key pair is $A = ((3, 391), (235, 391))$.
   $P(A) = (3, 391)$ is the public key.
   $S(A) = (235, 391)$ is the private key.
6. Bob gets $P(A)$ from Alice and encrypts M:

$$C = M^e \bmod n = 314^3 \bmod 391 = 155$$

7. Bob sends C to Alice.
8. Alice encrypts C:

$$C^d \bmod n = 155^{235} \bmod 391 = 314$$

   which is in fact Bob's message.

Euclid's Algorithm

Algorithm: Euclid's algorithm.

```
Euclid($a$, $b$) $\longrightarrow d$
    Input: $a$, $b$, integers
    Output: $d$, the greatest common divisor of $a$ and $b$
1   if $b = 0$ then
2       return $a$
3   else
4       return Euclid($b$, $a \bmod b$)
```

Example: `Euclid(160, 144)`

a	b	$a \bmod b$
160	144	16
144	16	0
16	0	

Extended Euclid's Algorithm

Algorithm: Extended Euclid's algorithm.

```
ExtendedEuclid($a, b$) $\longrightarrow (r, x, y)$
    Input: $a$, $b$, positive integers
    Output: $r$, $x$, $y$, such that $r = \gcd(a, b) = xa + yb$
1   if $b = 0$ then
2       return $(a, 1, 0)$
3   else
4       $(r, x, y)$ = ExtendedEuclid($b$, $a \bmod b$)
5       return $(r, y, x - \lfloor a/b \rfloor \cdot y)$
```

Example for $a = 3$, $b = 352$

a	b	$a \bmod b$	$(r, y, x - \lfloor a/b \rfloor \cdot y)$
3	352	3	$(1, -117, 1 - \lfloor 3/352 \rfloor \times (-117)) = (1, -117, 1)$
352	3	1	$(1, 1, 0 - \lfloor 352/3 \rfloor \times 1) = (1, 1, -117)$
3	1	0	$(1, 0, 1 - \lfloor 3/1 \rfloor \times 0) = (1, 0, 1)$
1	0		$(1, 1, 0)$

$$1 = 3 \times (-117) + 352 \times 1$$

Modular Inverse Algorithm

Algorithm: Modular inverse algorithm.

```
ModularInverse($a, n$) $\longrightarrow r$
    Input: $a$, $n$, positive integers; $n$ is the modulus
    Output: $r$, such that $r = a^{-1} \bmod n$, if it exists, or 0 otherwise
1   $(r, x, y)$ = ExtendedEuclid($a, n$)
2   if $r \neq 1$ then
3       return 0
4   else if $x < 0$ then
5       return $x + n$
6   else
7       return $x$
```

Equivalence of Encryption and Signature

- For two integers u, v, we have:

$$(u \bmod n)(v \bmod n) \bmod n = uv \bmod n$$

- So the decryption is:

$$M = D_{S(A)}(C) = C^d \bmod n = (M^e \bmod n)^d \bmod n = M^{ed} \bmod n$$

- The verification of the signature is:

$$D_{P(A)}(C) = C^e \bmod n = (M^d \bmod n)^e \bmod n = M^{de} \bmod n$$

RSA Correctness

The RSA correctness is based on the fact that it can be proved that:

$$M = M^{ed} \bmod n$$

Message Size Limit

- The encryption of a message M is:

$$C = E_{P(A)}(M) = M^e \bmod n$$

- Say that Alice wants to encrypt a message M.
- If p and q are each 2048 bits, then $p - 1$ and $q - 1$ are 2047 bits, so n will have length $2047 + 2047 = 4094$ bits.
- So M must have length up to 4094 bits. If it does not, Alice must break M in pieces up to 4094 bits each and encrypt them separately.

RSA Security

- The security of RSA, according to our current knowledge, is based on the difficulty of analyzing a number to its prime factors (factoring).
- If somebody intercepts n, there is no efficient way to find p and q, so that the message could be decrypted.
- One caveat: quantum computation.

Why not only RSA?

- As RSA solves the key exchange problem and is secure, why don’t we use it exclusively, and do without symmetric cryptography?
- Because it is much slower than symmetric cryptography systems like AES.
- For that reason it is used in a hybrid fashion, as a key exchange mechanism.

Hybrid Cryptography (1)

1. Alice, who wants to send a large message M to Bob, so that RSA would be slow, chooses a random key K that she will use with a symmetric cryptographic system like AES.
2. Alice encrypts K using Bob's public key:

$$C_K = E_{P(B)}(K)$$

3. Alice encrypts M with AES using as key K:

$$C_M = E_{AES(K)}(M)$$

4. Alice sends (C_K, C_M) to Bob.

Hybrid Cryptography (2)

5. Bob decrypts Alice's encrypted message using his private key:

$$K = D_{S(B)}(C_K)$$

6. Bob uses the key K that he recovered in the previous step to decrypt M:

$$M = D_{AES(K)}(C_M)$$

Fast Signing (1)

- We have similar speed issues when we want to sign large messages.
- For that reason, instead of signing the whole message, we sign a *fingerprint* (or *digest*) of the message, which is much smaller, e.g., 256 bits.
- To get the fingerprint, we run a function $h(M)$ whose output is such that it is very difficult to have two messages M and M' with $h(M) = h(M')$.
- In this way the fingerprint identifies the message.
- The function $h(M)$ is a *hash function*.

Fast Signing (2)

- It is impossible for $h(M)$ not to create identical fingerprints for all possible different messages (to avoid *collisions*).
- If the messages to sign are 10 Kbytes, they have 80,000 bits so there are 2^{80000} such messages.
- If $h(M)$ creates 256 bits fingerprints, it maps 2^{80000} messages to 2^{256} fingerprints.
- Even though in theory we would get a very large numbef of collisions, in practice we want collisions to be very rare.
- Such functions are called *collision-free hash functions.*
- Finding such functions is a serious matter.

Fast Signing (3)

1. Alice calculates M's digest:

$$H = h(M)$$

2. Alice signs M's digest:

$$C = E_{S(A)}(H)$$

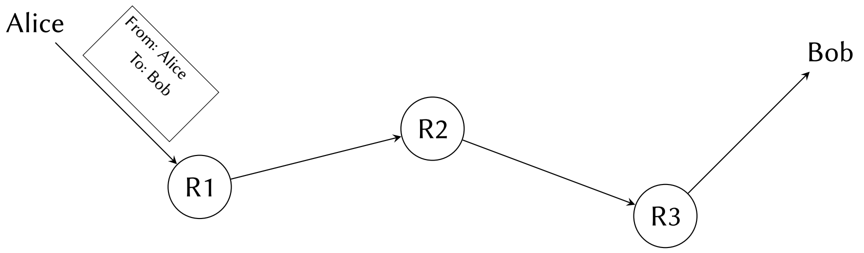
3. Alice sends $(M, C = E_{S(A)}(H))$ to Bob.
4. Bob calculates $H = h(M)$.
5. Bob verifies that the message has been signed by Alice:

$$H = D_{P(A)}(C)$$

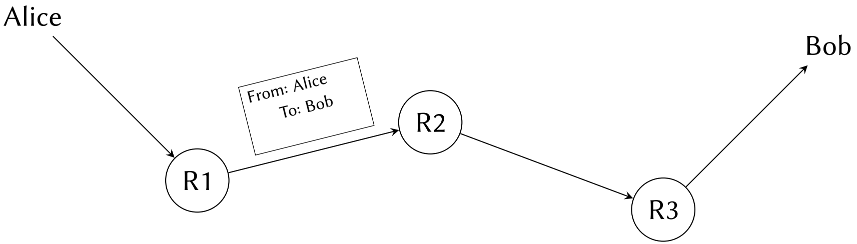
6. Bob verifies that the fingerprint he calculated in step 4 is the same with the signature he verified it is Alice's in step 5.

Outline

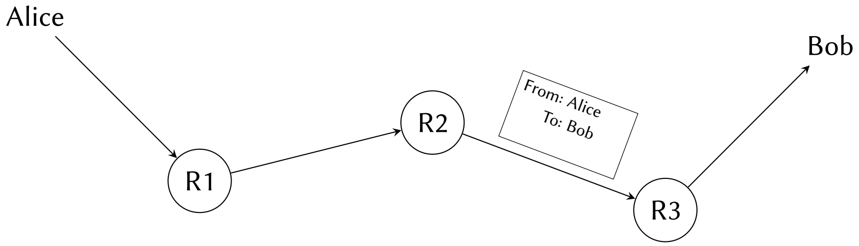
Message Traveling from Alice to Bob (1)



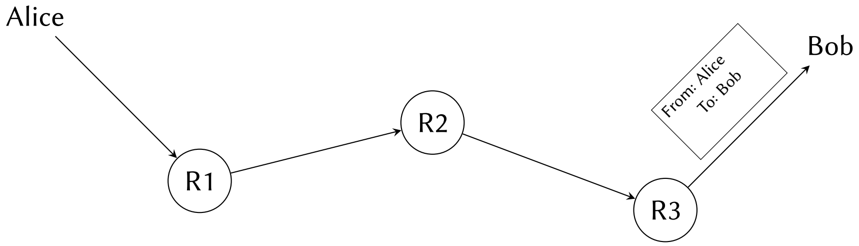
Message Traveling from Alice to Bob (2)



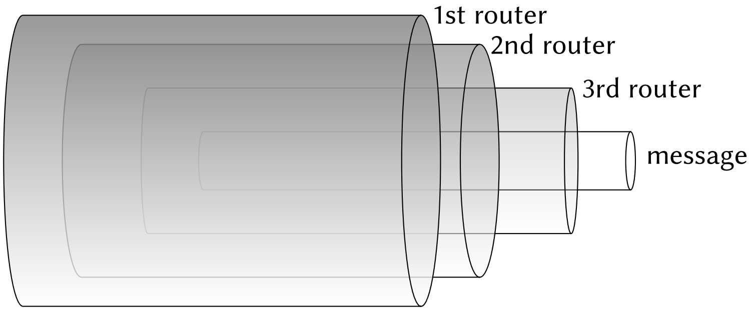
Message Traveling from Alice to Bob (3)



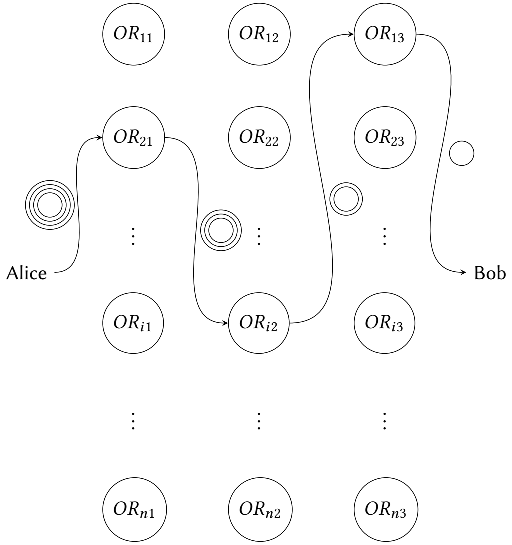
Message Traveling from Alice to Bob (4)



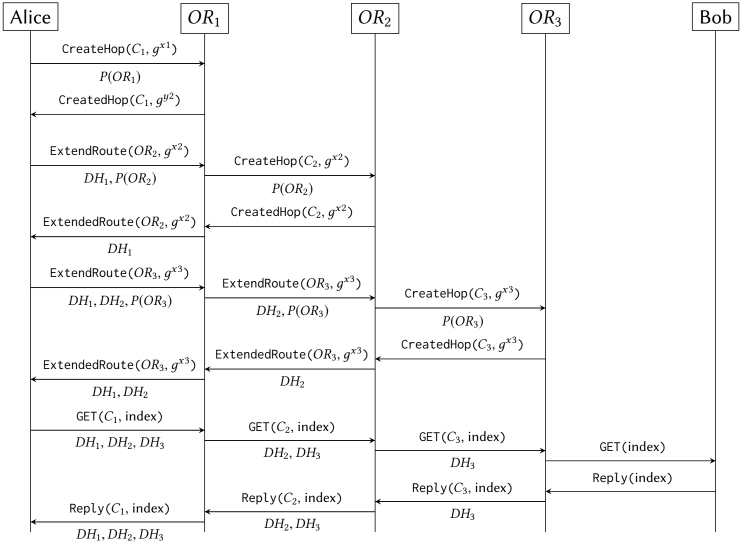
Onion Routing



TOR: The Onion Router



Onion Routing in Tor


Alice
OR_1
OR_2
OR_3
Bob
CreateHop(C_1, g^{x1})
$P(OR_1)$
CreatedHop(C_1, g^{y2})
ExtendRoute(OR_2, g^{x2})
$DH_1, P(OR_2)$
CreateHop(C_2, g^{x2})
$P(OR_2)$
CreatedHop(C_2, g^{x2})
ExtendedRoute(OR_2, g^{x2})
DH_1
ExtendRoute(OR_3, g^{x3})
$DH_1, DH_2, P(OR_3)$
ExtendRoute(OR_3, g^{x3})
$DH_2, P(OR_3)$
CreateHop(C_3, g^{x3})
$P(OR_3)$
CreatedHop(C_3, g^{x3})
ExtendedRoute(OR_3, g^{x3})
DH_2
ExtendedRoute(OR_3, g^{x3})
DH_1, DH_2
GET(C_1, index)
DH_1, DH_2, DH_3
GET(C_2, index)
DH_2, DH_3
GET(C_3, index)
DH_3
GET(index)
Reply(index)
Reply(C_3, index)
DH_3
Reply(C_2, index)
DH_2, DH_3
Reply(C_1, index)
DH_1, DH_2, DH_3