Chapter 9: Public Key Encryption

Information Security

Nguyễn Đăng Quang

Fall 2022

Goals

- Modular Arithmetic,
- RSA Encryption,
- Discrete logarithm
- Diffie - Hellman

Introduction to Modular Arithmetic

- **Modulo**

$$\frac{A}{B} = Q \text{ remainder } R$$

A is the dividend
B is the divisor
Q is the quotient
R is the remainder

R = A mod B say: A modulo B is equal to R where B is modulus

- **Congruent Modulo** $A \equiv B \pmod{C}$ A is congruent to B modulo C

Properties

Addition

(a + b) mod n = (a mod n + b mod n) mod n

Subtraction

(a - b) mod n = (a mod n - b mod n) mod n

Multiplication

(a * b) mod n = (a mod n * b mod n) mod n

Exponentiation

$$a^x \bmod n = (a \bmod n)^x \bmod n$$

Euler's totient function

$\varphi(n)$: (Euler Phi function) – the number of integers smaller than n and relatively prime (coprime) to n

- Ex: $\varphi(9)$ has 6 relatively prime to n: 1, 2, 4, 5, 7, 8

- If p is prime, $\varphi(p) = p-1$

- If $n = p \times q$ and p, q are primes, $\varphi(n) = (p-1)\times(q-1)$
  - Ex: Find $\varphi(21)$: $21 = 3\ (p) \times 7\ (q) \Rightarrow \varphi(21) = (3-1) \times (7-1) = 12$

Euler’s Theorem

If gcd(a,n) = 1 then

$$a^{\varphi(n)} \equiv 1\ (mod\ n)$$

Example:

φ(10)=4, so if gcd(a,10) = , then $a^4 \equiv 1$ (mod 10)

Extended Euclidean Algorithm

- GCD(a,b): a*x + b*y = gcd(a,b)
- If gcd(a,b) = 1 (a,b are coprime) ➔ mod b for both side:
- a*x = 1 (mod b) → x is the modular inverse of a

Modular inverse

- The modular inverse of A (mod C) is A^{-1}
- $(A * A^{-1}) \equiv 1 \pmod{C}$ or equivalently $(A*A^{-1}) \bmod C = 1$
- Only the numbers coprime to C have a modular inverse (mod C)

Example: Find modular inverse for A (mod C):

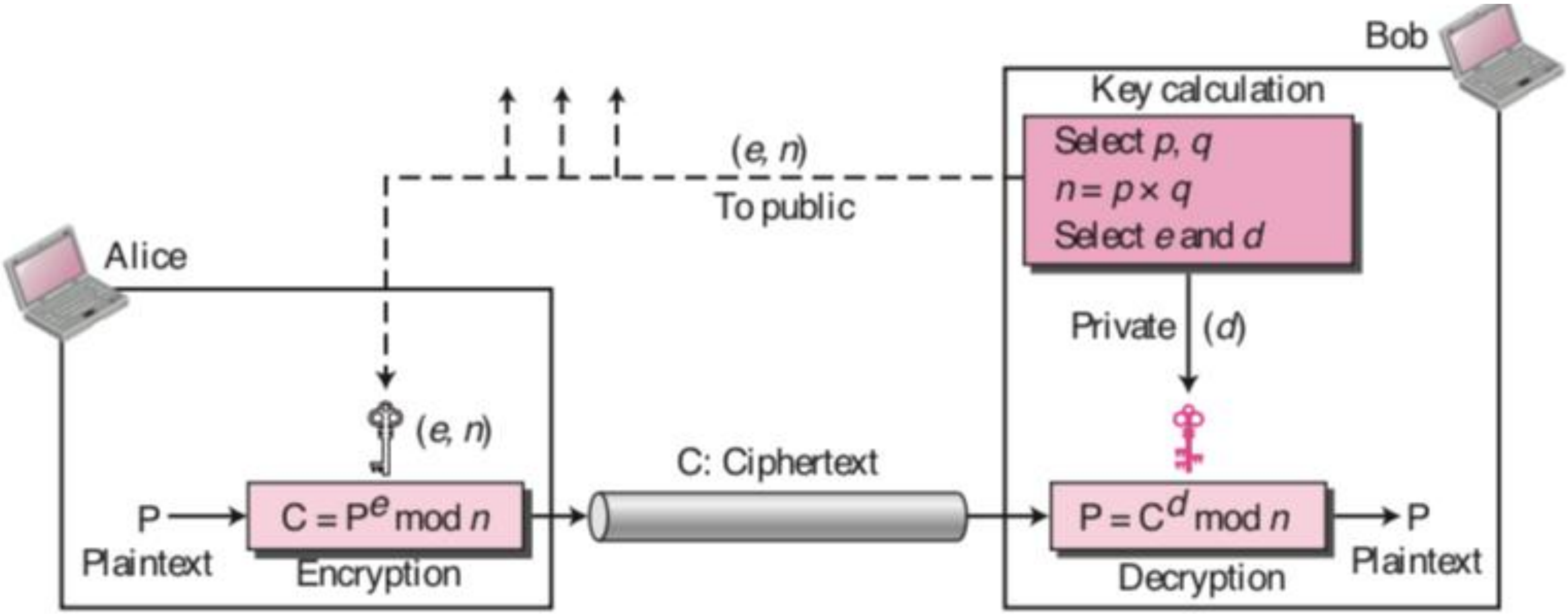
for m in range©: if A * m mod C = 1 ➔ m is modular inverse of A (mod C)

$3 * 0 \equiv 0 \pmod{7}$
$3 * 1 \equiv 3 \pmod{7}$
$3 * 2 \equiv 6 \pmod{7}$
$3 * 3 \equiv 9 \equiv 2 \pmod{7}$
$3 * 4 \equiv 12 \equiv 5 \pmod{7}$
$3 * 5 \equiv 15 \pmod{7} \equiv \underline{1} \pmod{7}$ <------ FOUND INVERSE!
$3 * 6 \equiv 18 \pmod{7} \equiv 4 \pmod{7}$

RSA

- Named after its inventors (Rivest, Shamir, Adleman).
- RSA is the most widely used public key algorithm, supports both public key encryption and digital signature.
- The security strength of RSA is based on the hypothesis that, factoring a very large number into two primes is a very hard problem.

Encryption, Decryption, and key generation in RSA



RSA Algorithm

- Select two prime numbers, p and q
- Compute RSA modulus n = p x q
- Compute φ(n) = (p-1) x (q-1) (2048 bits)
- Select an integer e that is relatively prime to φ(n)
- Find d which is modular inverse of e mod φ(n).
- The public key is (e, n)
- The private key is (d, n)

Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calcuate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption by Alice with Alice's Public Key	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

Example

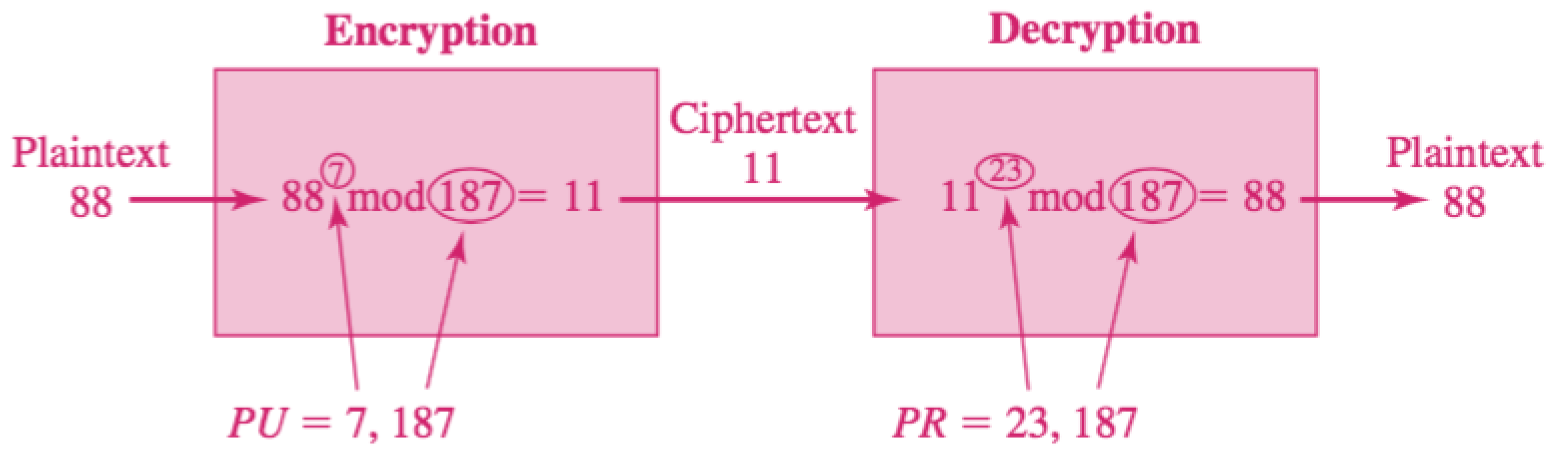
- Select two prime numbers, $p = 17$ and $q = 11$.
- Calculate $n = p*q = 17 * 11 = 187$.
- Calculate $\varphi(n) = (p - 1)(q - 1) = 16 * 10 = 160$.
- Select e relatively prime to $\varphi(n) = 160$ and less than $\varphi(n)$ → $e = 7$.
- Determine d such that $de \equiv 1 \pmod{160}$ and d 6 160. The correct value is $d = 23$, because $23 * 7 = 161 = (1 * 160) + 1$;
- Public key PU = {7,187}
- Private key PR = {23,187}

Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calcuate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

RSA Encryption & Decryption

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption by Alice with Alice's Public Key	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$



Step-by-step encryption process

Convert the message "Hello world" into ASCII values:

"H" = 72

"e" = 101

"l" = 108

"l" = 108

"o" = 111

" " (space) = 32

"w" = 119

"o" = 111

"r" = 114

"l" = 108

"d" = 100

Encrypt each value using public key {7,187}

72 → 72^7 mod 187 = 1,028,071,702 mod 187 = 66

101→ 101^7 mod 187 = 10,201,010,101 mod 187 = 128

108→ 108^7 mod 187 = 1,782,969,984 mod 187 = 121

108→ 108^7 mod 187 = 1,782,969,984 mod 187 = 121

111: 111^7 mod 187 = 2,487,388,671 mod 187 = 49

32→ 32^7 mod 187 = 1,073,741,824 mod 187 = 1

119→ 119^7 mod 187 = 1,872,517,119 mod 187 = 119

111→ 111^7 mod 187 = 2,487,388,671 mod 187 = 49

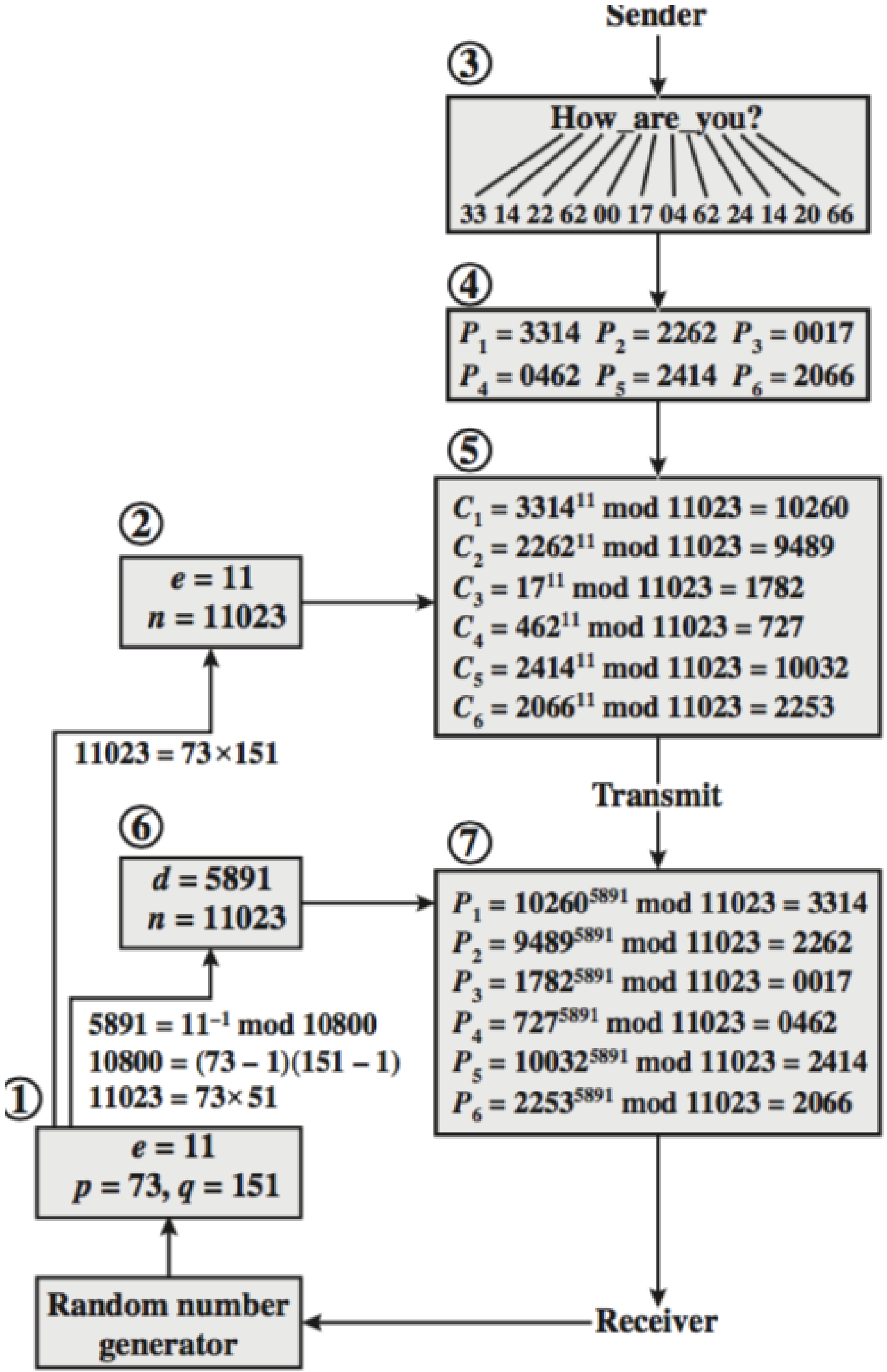
114→ 114^7 mod 187 = 3,972,969,984 mod 187 = 161

108→ 108^7 mod 187 = 1,782,969,984 mod 187 = 121

100→ 100^7 mod 187 = 1,000,000,000 mod 187 = 100

RSA Example

RSA processing of multiple blocks



Quiz

1. Given $p = 3$, $q = 11$
2. Compute $n = ?$
3. Compute $\varphi(n) = ?$
4. Assume $e = 7$, compute $d = ?$
5. The public key $(e, n) = ?$
6. The private key $(d, n) = ?$
7. Suppose $m = 2$, what is the encryption of m. $Enc(m) = ?$
8. Check that the decryption of $Enc(m)$ equals to m ?

Tools

Generate RSA keys: openssl genrsa –aes128 –out private.pem 1024

View the Private key: openssl rsa –in private.pem –noout –text

View keys in text: openssl rsa –in private.pem –text

Extract the Public key: openssl rsa –in private.pem –pubout > public.pem

View: openssl rsa –in public.pem –pubin –text

Encrypt & Decrypt

Encrypt: openssl rsautl –encrypt –inkey public.pem –pubin –in msg.txt –out msg.enc

Decrypt: openssl rsautl –decrypt –inkey private.pem –in msg.enc

Performance measurement

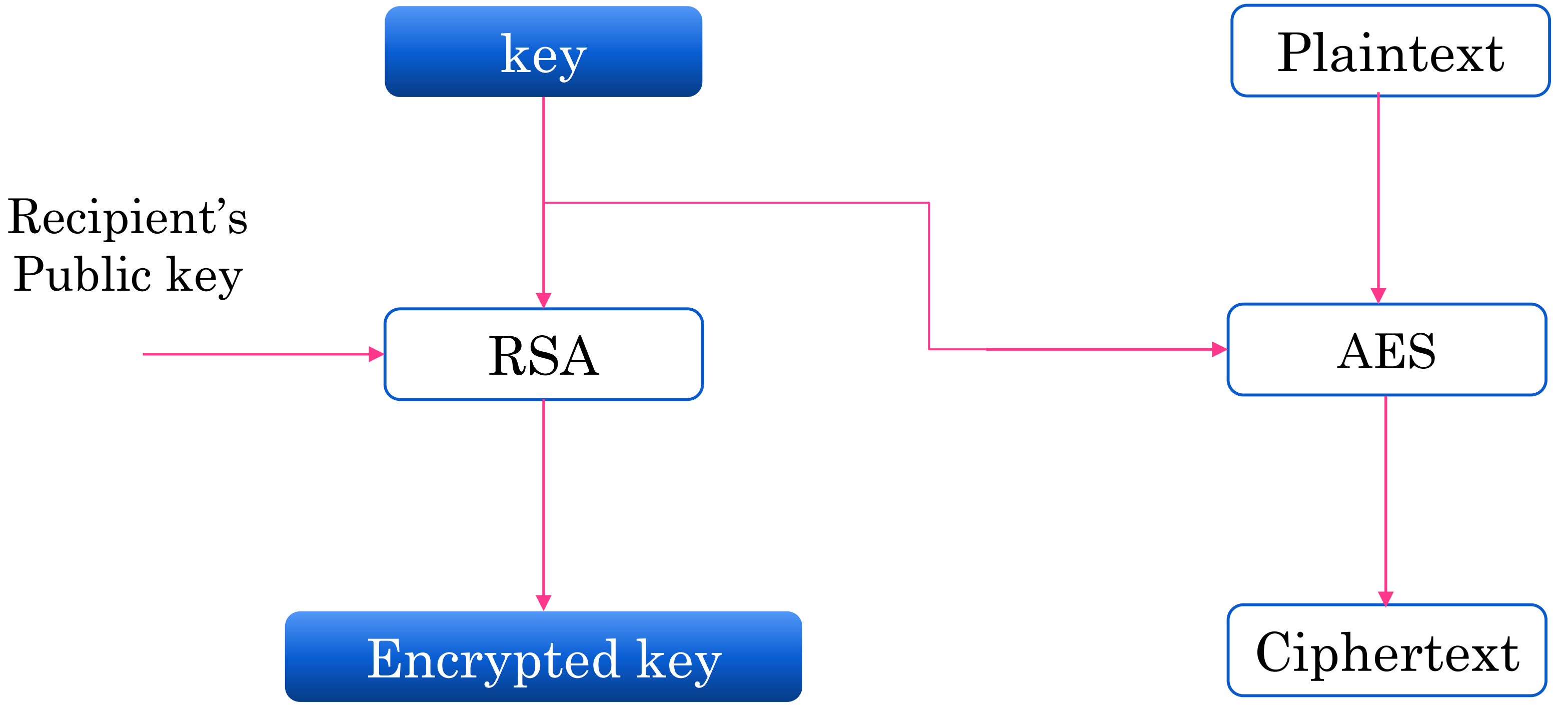
Strength:

- 1024-bit RSA key = 80-bit symmetric key
- 2048-bit RSA key = 112-bit symmetric key
- 3072-bit RSA key = 128-bit symmetric key

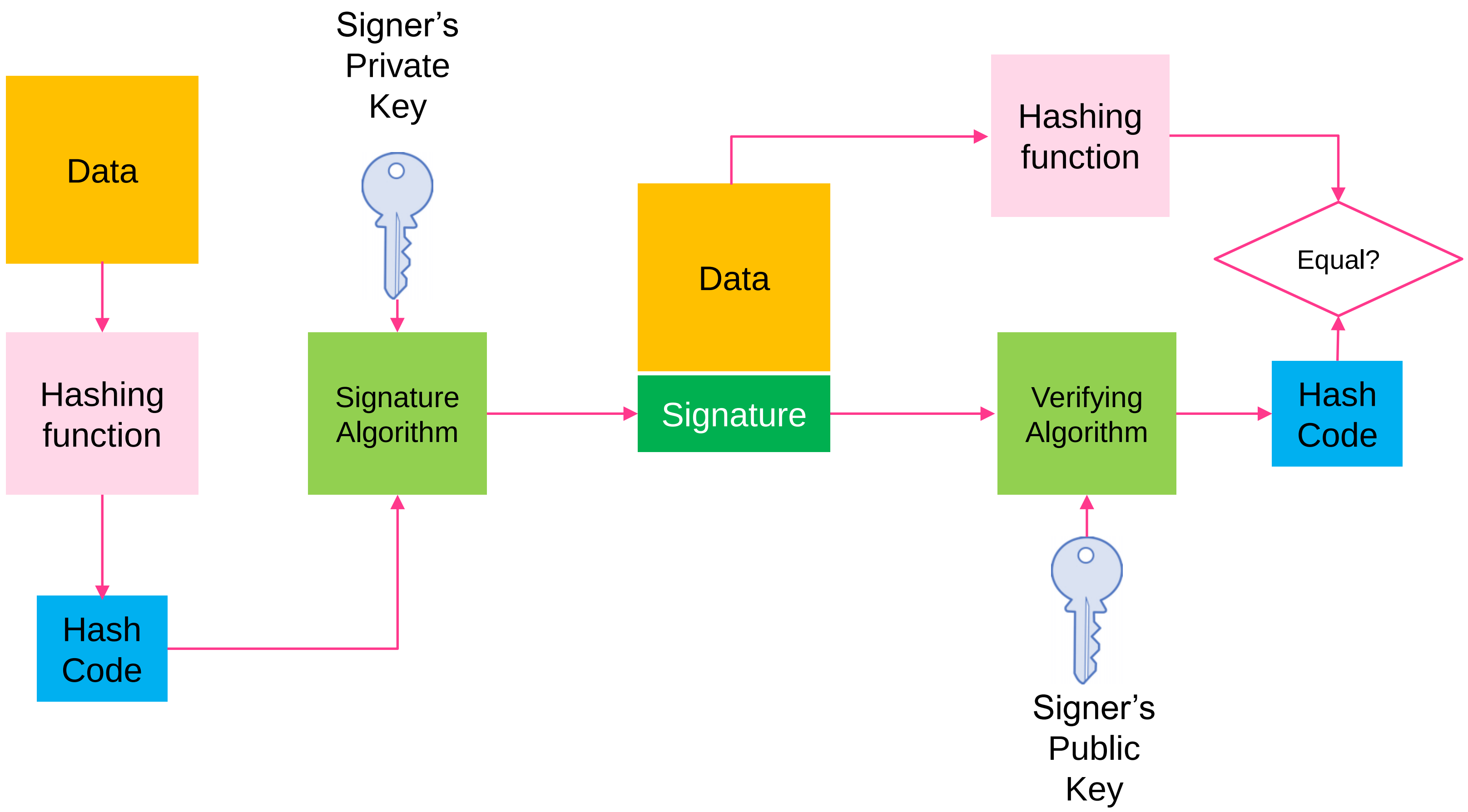
openssl speed rsa

openssl speed aes-128-cbc

Hybrid Encryption



Digital Signature



Digital signature with Openssl

- Generating hash

  openssl sha256 –binary msg.txt > msg.sha256

- Signing and Verifying

  Signing:

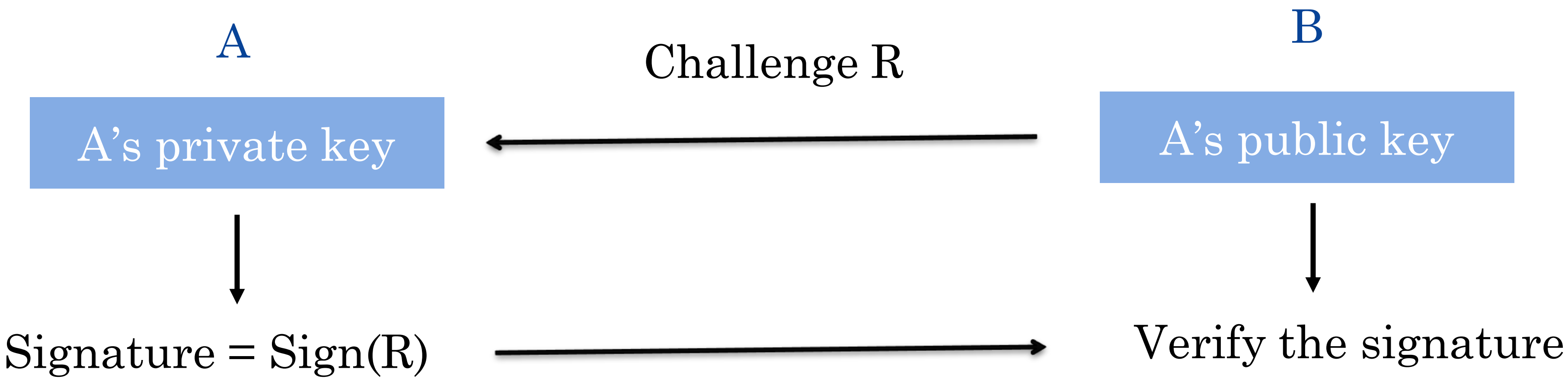
  openssl rsautl –sign –inkey private.pem –in msg.sha256 –out msg.sig

  Verify the signature:

  openssl rsautl –verify –inkey public.pem –in msg.sig –pubin –raw |xxd

Other applications

Public-key based Authentication



Github SSH keys

SSH keys

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

Authentication Keys

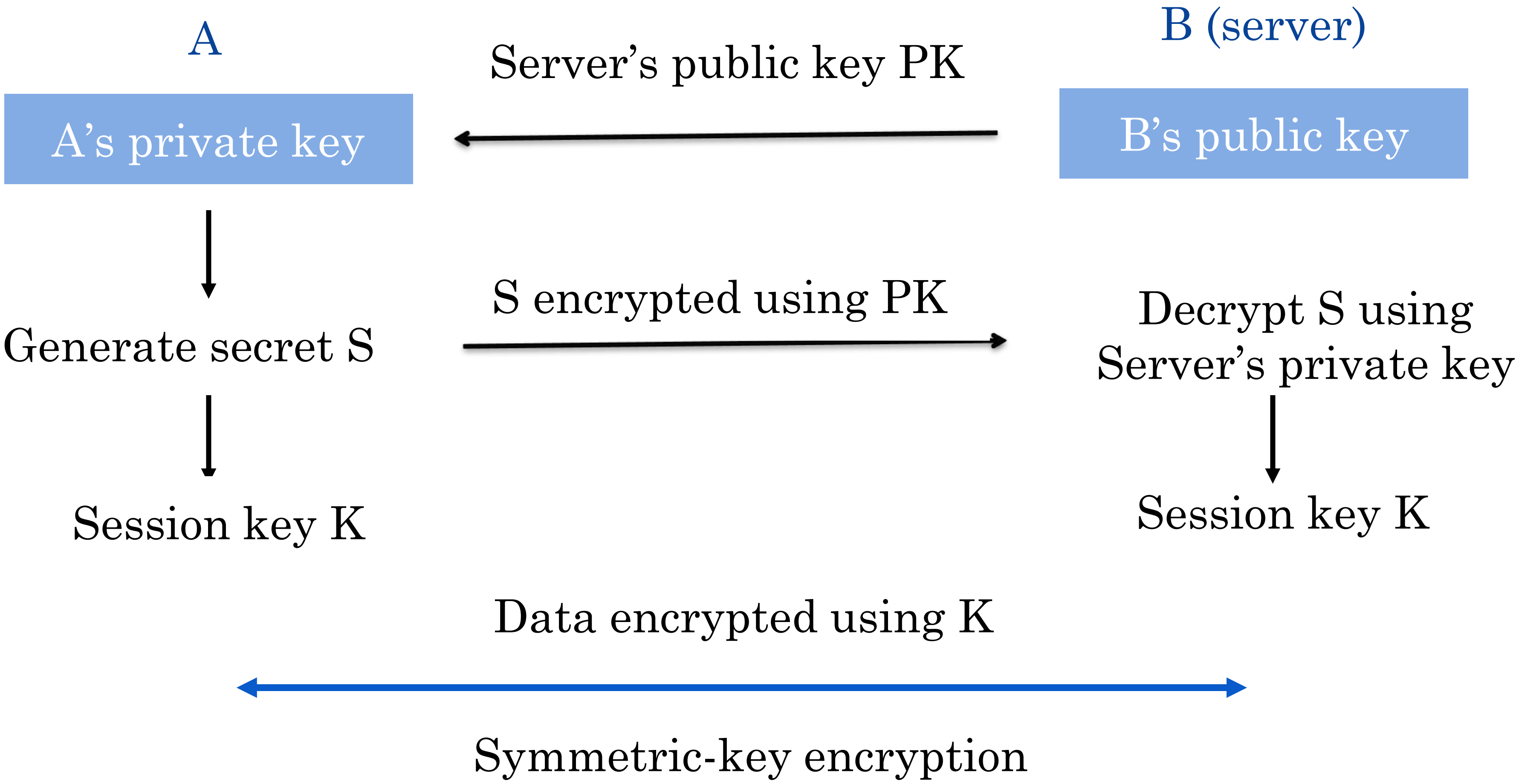
SSH

Mac mini M1
`SHA256:YH3HGT5/Y98WnTJ7jJ1yRidXWuZUS9FHOU4oprFEV5k`
Added on Nov 27, 2022
Never used — Read/write

Delete

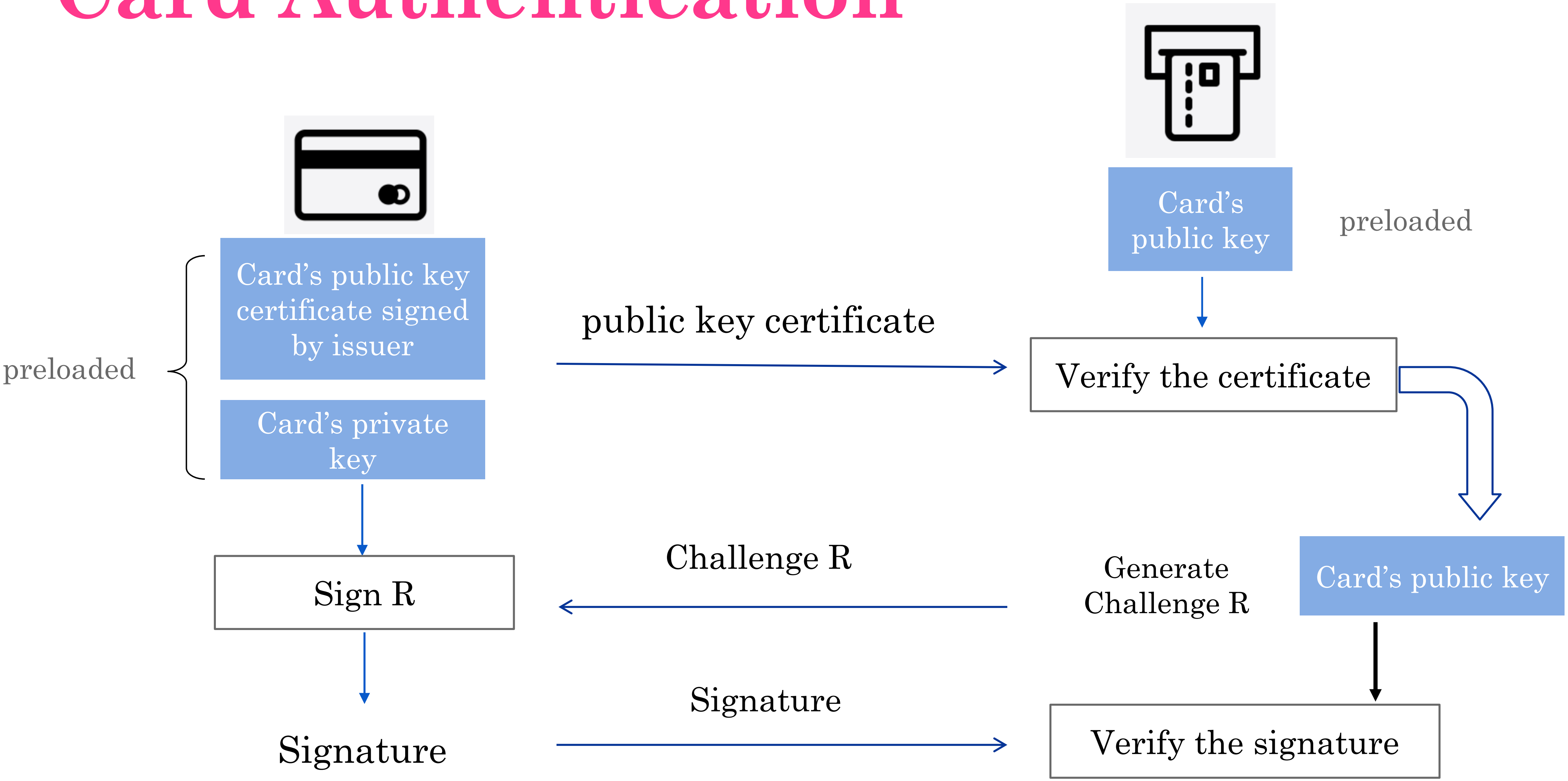
Check out our guide to generating SSH keys or troubleshoot common SSH problems.

HTTPs

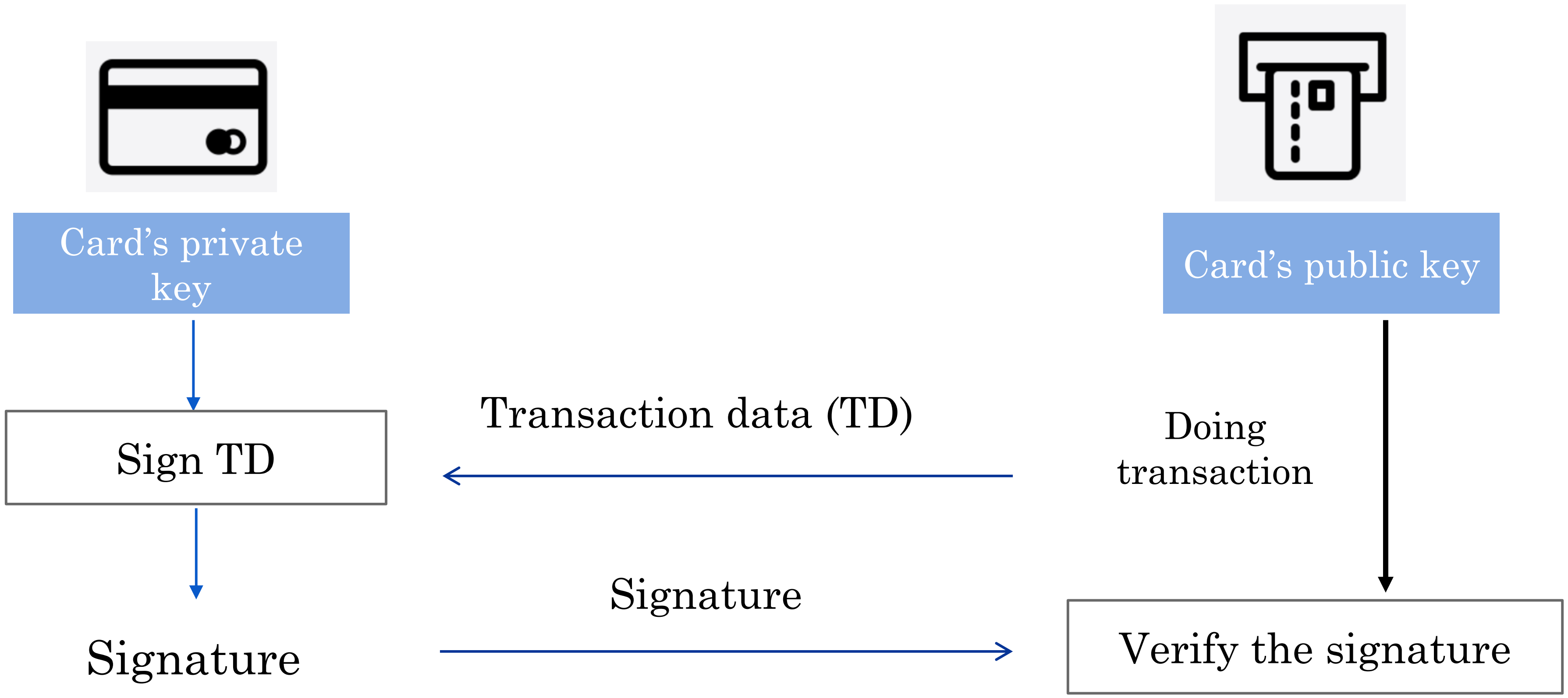

A
B (server)
Server's public key PK
A's private key
B's public key
S encrypted using PK
Generate secret S
Decrypt S using
Server's private key
Session key K
Session key K
Data encrypted using K
Symmetric-key encryption

Credit Cards

Card Authentication



Transaction Authentication



Diffie-Hellman Key Exchange

- First published public-key algorithm.
- By Diffie and Hellman in 1976 along with the public key concepts.
- Used in a number of commercial products.
- Practical method to exchange a secret key securely that can be used for subsequent encryption messages.
- Security relies on difficulty of computing discrete logarithm.

Recall…

Arithmetic

- $y = 2^x$: exponent
- $x = \log_2 y$: logarithm (calculate the power x)

Modular arithmetic

(modulus p)

- $y \equiv 2^x \pmod{p}$
- $x \equiv \log_2 y \pmod{p}$

Discrete logarithm

- Let p: the prime modulus
- Let g: the primitive root of p
- Calculate $y = g^x \bmod p$, the result are all numbers in range 1→p-1
- Example: p = 11 ➔ g = 2,

```
for x in range(1, p):
        g = 2**x
        k = g % p
        print(k, end=',') ➔ 2, 4, 8, 5, 10, 9, 7, 3, 6, 1
                            (all numbers in range 1 → 11)
```

- g is also called the generator
- Calculate x from y is the discrete logarithm problem. If p is chosen as a very long number, the time to calculate x is extremely long.

Diffie and Hellman Key Exchange

- In the **Diffie-Hellman protocol** two parties create a symmetric session key without the need of a Key Distribution Center (KDC);
- The two parties need to choose two numbers p and g;
- p is a prime modulus, g is a generator
- These two numbers do not need to be confidential. They can be sent publicly through the Internet;

Key Exchange protocol steps

1. Alice chooses a large random number x $(0 \leq x \leq p - 1)$ and calculates R1 = $g^x \bmod p$.

2. Alice sends R1 to Bob

3. Bob chooses another large random number y $(0 \leq y \leq p - 1)$ and calculates R2 $= g^y \bmod p$.

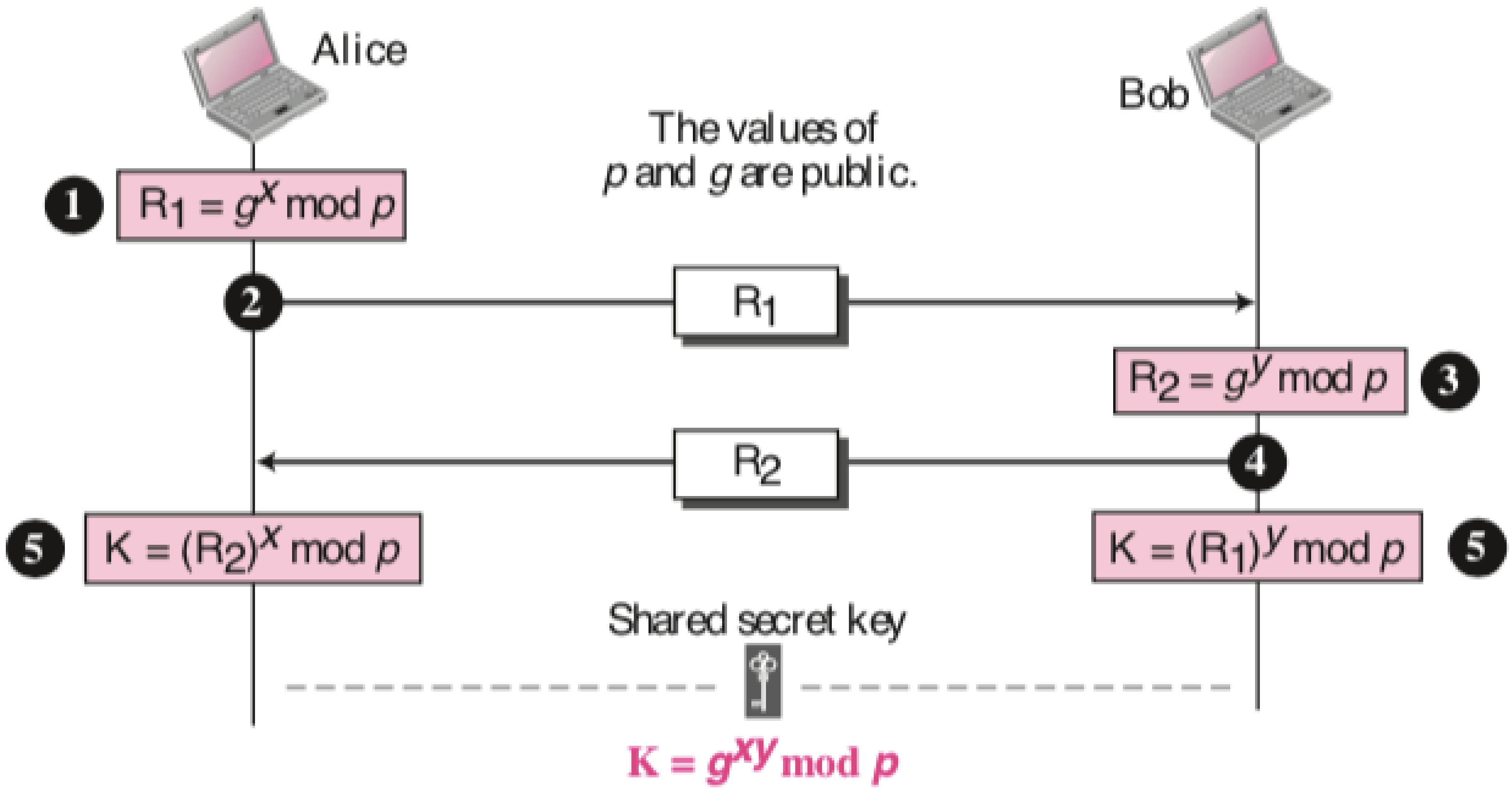
4. Bob sends R2 to Alice

5. Alice calculates K = $(\mathrm{R2})^x \bmod p$. Bob also calculates K = $(\mathrm{R1})^y \bmod p$. K is the symmetric key for the session

$$\left.\begin{aligned} &\text{Alice: } (\mathrm{R2})^x \bmod p = (g^y \bmod p)^x \bmod p = (g^y)^x \bmod p = KA \\ &\text{Bob: } (\mathrm{R1})^y \bmod p = (g^x \bmod p)^y \bmod p = (g^x)^y \bmod p = KB \end{aligned}\right\} \quad KA=KB$$

Symmetric-Key Agreement

Diffie-Hellman Key Agreement



Turn DH to public-key encryption

1. Alice & Bob agree on g,p

2. Alice generates (public, private) key-pair: (g, p, g^x mod p), x.

   the public-key (g, p, g^x mod p) *is sent to Bob*

3. Bob computes (g^x mod p)^y mod p = g^{xy} mod p *which is the common key to decrypt*