Asymmetric Cryptography

Introduction to Security (192.019)

Mauro **Tempesta**

Security & **Privacy** Research Unit (192-06)
https://**secpriv.wien**

Key Management

Symmetric cryptography

- N communication parties require $N \cdot (N-1)/2$ secret keys to communicate with each other
  - Each party needs $N-1$ keys that must be shared in a secure way
  - This process must be repeated every time the key is refreshed

Asymmetric cryptography

- N communication parties require N key pairs to communicate with each other
  - Every party needs their own key pair and the public keys of the others
  - Public keys can be distributed over insecure channels (e.g., the Internet)

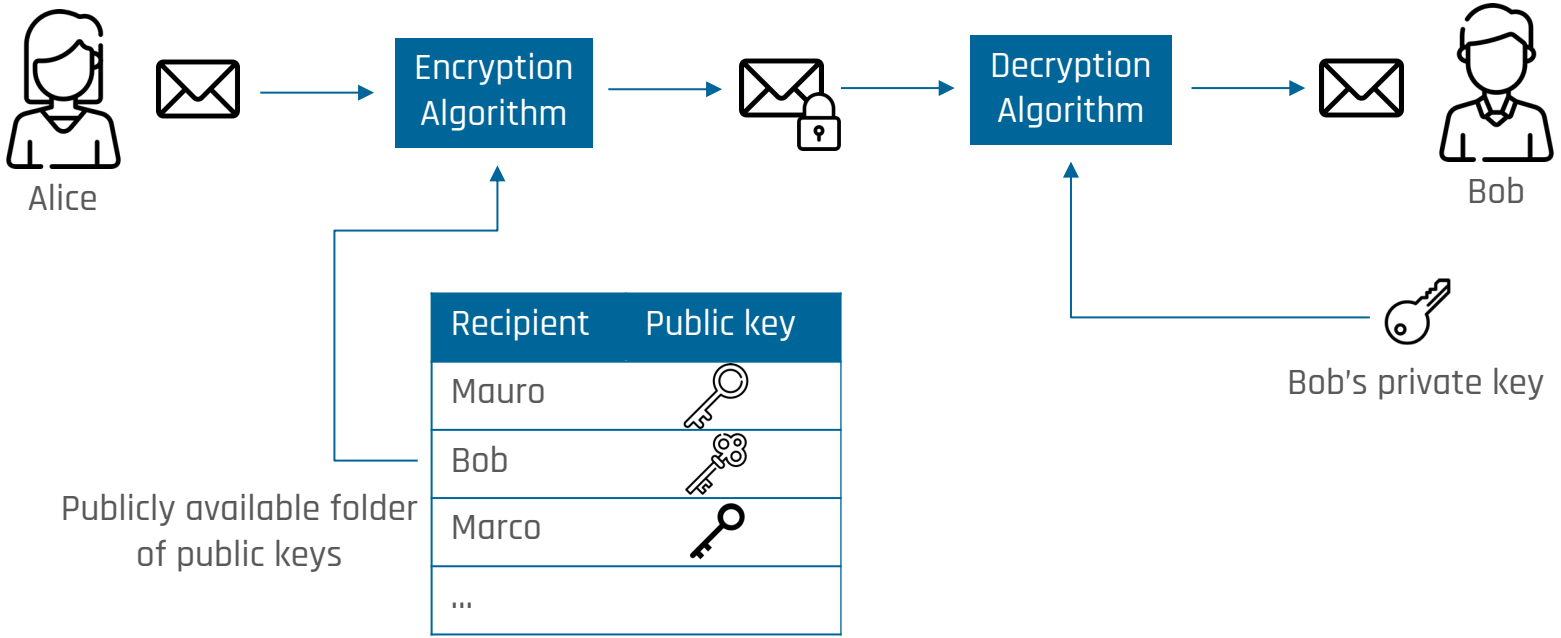
Asymmetric Encryption

Asymmetric Encryption

- Asymmetric encryption schemes use **different keys** for encryption and decryption
  - The decryption key **must not** be derivable from the encryption key
  - This holds even if the attacker possesses pairs of plaintext and corresponding ciphertexts (**known-plaintext attack**)

- If this condition holds, there is **no reason** to keep the encryption key secret
  - Encryption key = **public key**
  - Decryption key = **private key**

Usage of an asymmetric encryption scheme



One-Way Functions in Asymmetric Schemes

- Asymmetric schemes use specific classes of **one-way functions** to satisfy the previously mentioned constraint about keys

- **Integer factorization**
  - Given two large prime numbers, it is easy to compute their product
  - It is **infeasible** to compute the factorization of a number made of large primes
  - Security of **RSA** is based on the difficulty of integer factorization

- **Discrete logarithms**
  - It is easy to compute $g^x \bmod p$ from g, x, p
  - It is **infeasible** to compute x from $g, p, g^x \bmod p$
  - Security of **DSA**, **ElGamal**, and the **Diffie-Hellman protocol** is based on this

The RSA Algorithm

Mathematical Background

- The **Euler's totient function** ϕ maps each integer n to the number of integers up to n that are **coprime** to n (i.e., greatest common divisor is 1)
  - If n is prime, $\phi(n) = n - 1$
  - If $n = pq$ (with p, q different primes):

$$\phi(n) = pq - (p-1) - (q-1) - 1 = (p-1)(q-1)$$

Euler's theorem

Let a, n be **coprime**, positive integers: then $a^{\phi(n)} \equiv 1 \pmod{n}$

RSA

- Published in 1977 by R. Rivest, A. Shamir, L. Adleman
  - Can be used for encryption, digital signatures and key management

- Let N be the **modulus** consisting of two large primes p, q
  - Public key (e, N) where e is the **public exponent**
  - Private key (d, N) where d is the **private exponent**

- Security depends on the size of p, q
  - Primes are typically between **1024** and **2048 bits** long (1024 bits $\approx 10^{300}$)
  - The larger, the better (but also slower!)

Key Generation

- Generate two **large primes** p, q
- Compute the modulus $N = pq$
- Choose a public exponent $e < N$ **coprime** to $\phi(N)$
  - To speed-up encryption, e is typically a small number (e.g., $65537 = 2^{16} + 1$)
- The private exponent d is the **modular inverse** of e modulo $\phi(N)$
  - If the inverse exists, $ed \equiv 1 \pmod{\phi(N)}$
  - The **extended Euclidean algorithm** can be used for this purpose

Generation of Large Prime Numbers

- Randomly generate an odd number of the desired bit length
- Test if small prime numbers (e.g., those smaller than 1000) divide the generated number
- Iterate the **Miller-Rabin test** on the generated number to ensure primality
  - If the test says that the number is not prime, the answer is **always correct**
  - If the test says that the number is prime, the answer **might be wrong** (probability < 1/4)
  - Iterating the test r times reduces the probability of mistakes to $\mathbf{1/4^r}$

Encryption and Decryption (Textbook RSA)

- Let $m < N$ be the plaintext and c the corresponding ciphertext

Encryption

$$c = m^e \bmod N$$

Decryption

$$m = c^d \bmod N$$

Correctness of RSA

For all positive, integer messages $m < N$, it holds that $m^{ed} \equiv m \ (mod\ N)$

$$
\begin{aligned}
& m^{ed} \bmod N \\
&= \ m^{k\phi(N)+1} \bmod N && \text{By construction of } d\text{, for some } k \geq 0 \\
&= m\left(m^{\phi(N)}\right)^{k} \bmod N && \text{Laws of exponents}
\end{aligned}
$$

Case $\gcd(m, N) = 1$

$$
\begin{aligned}
& m\left(m^{\phi(N)}\right)^{k} \bmod N \\
&= m \cdot 1^{k} \bmod N && \text{Euler's theorem} \\
&= m
\end{aligned}
$$

Correctness of RSA

For all positive, integer messages $m < N$, it holds that $m^{ed} \equiv m \ (mod\ N)$

Case $\gcd(m, N) > 1$

- Since $m < N$, either $\gcd(m, N) = p$ or $\gcd(m, N) = q$
  - We assume $\gcd(m, N) = p$, the other case is similar
  - Since $\gcd(m, q) = 1$, we have $m^{\phi(q)} \bmod q = 1$ (Euler's theorem)

$$
\begin{aligned}
& m^{\phi(N)} \bmod q \\
&= m^{\phi(p)\phi(q)} \bmod q && \text{Multiplicative property of } \phi \\
&= (m^{\phi(q)})^{\phi(p)} \bmod q && \text{Laws of exponents} \\
&= 1^{\phi(p)} \bmod q = 1 && \text{Euler's theorem}
\end{aligned}
$$

Correctness of RSA

For all positive, integer messages $m < N$, it holds that $m^{ed} \equiv m \ (mod\ N)$

Case $\gcd(m, N) > 1$

1. Since we assumed $\gcd(m, N) = p$, we have $m = jp$ for some $j \geq 0$
2. Since $m^{\phi(N)} \bmod q = 1$, we have $(m^{\phi(N)})^a \bmod q = 1$ for all $a \geq 0$

$$
\begin{aligned}
& m\left(m^{\phi(N)}\right)^k \bmod N \\
&= m(wq + 1) \bmod N && \text{Property 2 above and definition of the modulo operation} \\
&= (wqjp + m) \bmod N && \text{Property 1} \\
&= (wjN + m) \bmod N = m && \text{Definition of modulus}
\end{aligned}
$$

Security of RSA

- Security of RSA depends on the **difficulty to factorize** the modulus N
  - If the prime factors p, q are known, the private exponent d can be **easily computed**
  - Factorization of "small" integers is available in public databases, e.g., http://factordb.com

- The secrecy of the totient value $\phi(N)$ is **equally crucial**
  - If leaked, the primes can be recomputed by solving the following system of equations

$$\begin{cases} N = pq \\ \phi(N) = (p-1)(q-1) \end{cases}$$

Issues of Textbook RSA for Encryption

- **Determinism** – If the same message is encrypted multiple times with the same public key, the produced ciphertexts are the same

- **Small messages and public exponents**
  - Given a ciphertext $c = m^e \bmod N$, it holds that $c + kN = m^e$ for some $k \geq 0$
  - When both m and e are small, m^e is **not much larger** than N
  - The plaintext m can be reconstructed by **brute-forcing** the possible values of k and computing the e^{th} root

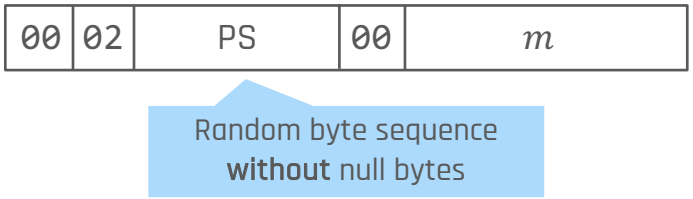
$$m = \sqrt[e]{c + kN}$$

Padding Algorithms for RSA Encryption

- The plaintext m is extended with **randomly generated components** to bring it to the size of the modulus N
  - The padded plaintext is then encrypted as usual
  - Usage of padding prevents **both** previous attacks

- Existing padding algorithms
  - **PKCS#1 v. 1.5**: simple, but **vulnerabilities** are known
  - **OEAP**: recommended for new applications

Padding with PKCS#1 v. 1.5



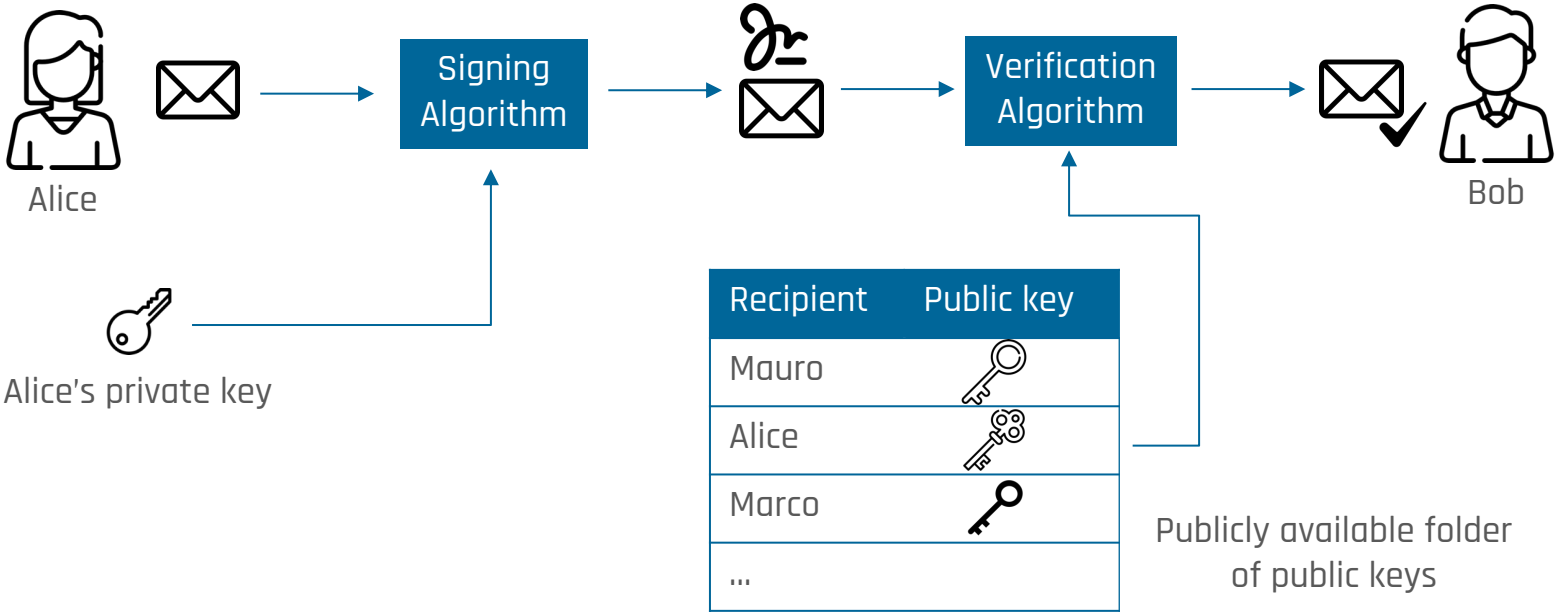
Digital Signatures

The Need of Digital Signatures

- Message Authentication Codes provide **data integrity** and **authenticity**
  - Can **only** be verified by the owners of the **secret key**
  - Do not offer **nonrepudiation**, since both sender and receiver can compute a valid MAC

- **Digital signatures** are asymmetric schemes that provide **data integrity**, **authenticity**, **public verifiability** and **nonrepudiation**
  - The **private key** is used to **compute** the signature
  - The **public key** is used to **verify** its validity

Usage of an asymmetric encryption scheme



Textbook RSA for Digital Signatures

- Let $m < N$ the message to be signed and s the corresponding signature

Signing

$$s = m^d \bmod N$$

Verification

Check if $s^e \bmod N$ equals m

Attacks against Textbook RSA for Digital Signatures

- **Malleability** of RSA can be used to generate **valid signatures** for new messages starting from signatures of other messages
  - Let $s_1 = m_1^d \bmod N$ and $s_2 = m_2^d \bmod N$ two valid signatures
  - $s_1 \cdot s_2 \bmod N = (m_1 \cdot m_2)^d \bmod N$ is the signature of the message $m_1 \cdot m_2$

- If the private key's owner is willing to sign **only** some of the attacker's messages, arbitrary signatures can be computed
  - Attacker wants to compute the signature of message m
  - Asks for the signature s' of message $m' = m \cdot r^e \bmod N$ (with $0 < r < N$)
  - The signature of m is $s = s' \cdot r^{-1} \bmod N$

$$s^e \, mod \, N = \left(s'^e \cdot r^{-e}\right) mod \, N = m' \cdot r^{-e} \, mod \, N = m \cdot r^e \cdot r^{-e} \, mod \, N = m$$

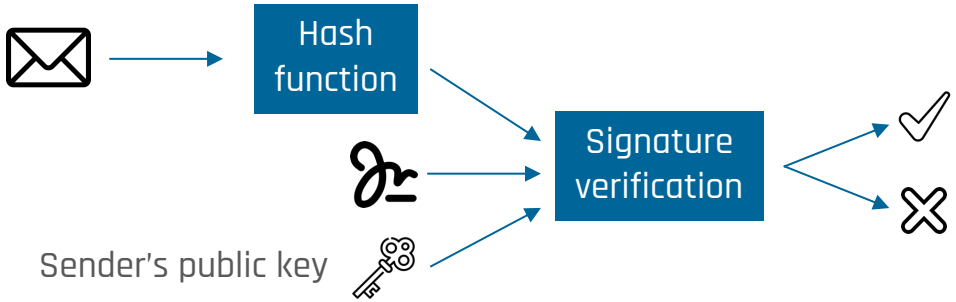
Further Issues with Textbook RSA

- Only messages **smaller** than the modulus N can be signed
  - One could split the message in multiple blocks and sign the individual parts

- Problems with the splitting approach
  - **Very slow** (one modular power per block)
  - Signature **as long as** the message
  - Missing connection between the individual signatures
    - Message blocks and the corresponding signature blocks could be swapped without problems (as in ECB mode): **no integrity**!

Hash Functions in Digital Signatures

- Sign the **hash value** of a message instead of the message itself
  - **Efficient** for arbitrarily large messages: a single hash computation (fast) and a single signing operation are required
  - Signature depends on **every bit** of the original message (**diffusion** of the hash function)
  - **Collision resistance** ensures that it is infeasible to find other messages for which the signature is valid



Security of Hash-and-Sign

- Previous attacks based on malleability are **prevented**
  - The product of the hash values of two messages is **not related** in any way to the product of the messages
  - Finding a message for which the product is a valid signature would amount to breaking the **one-way property** of the hash function

- Hash is usually **padded** before signature computation
  - PKCS#1 v. 1.5 (for signatures) and SSA are both considered secure
  - Padding helps to detect if the signature **has been tampered with**

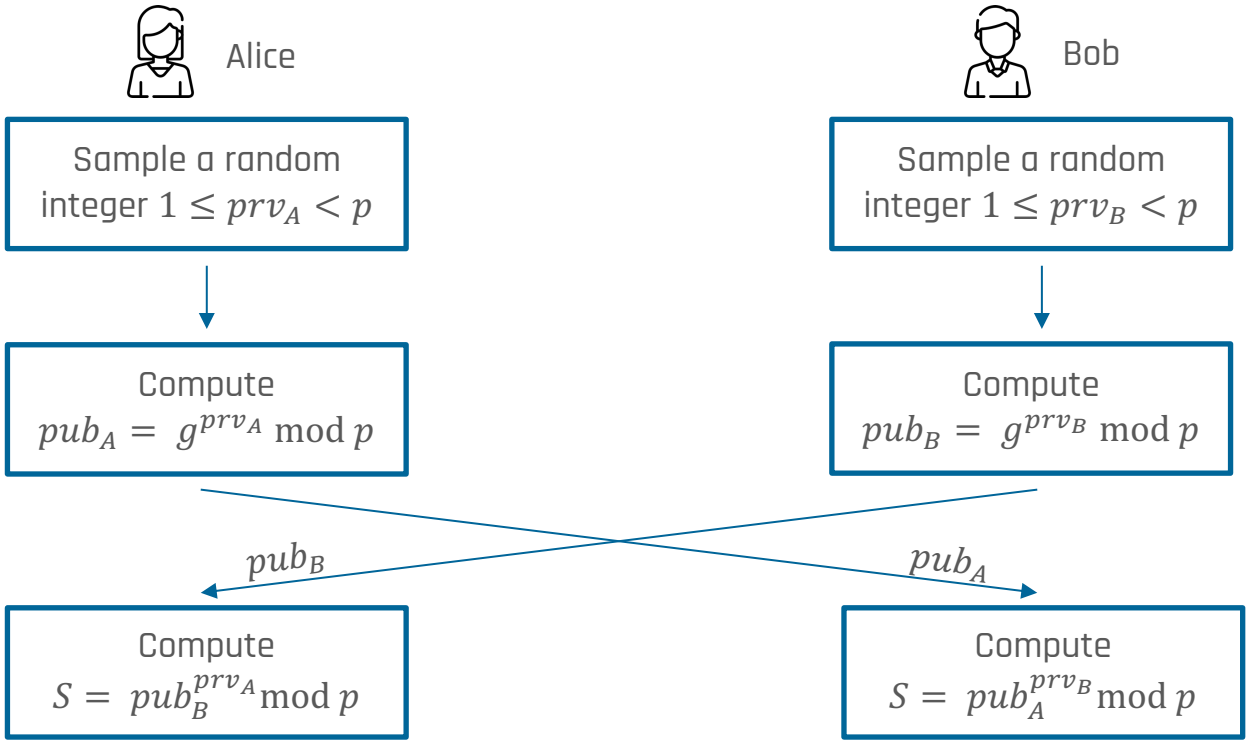
Diffie-Hellman Protocol

Diffie-Hellman Protocol

- Key exchange protocol that enables the generation of a **secret** shared between two communicating parties over a **public channel**
  - First practical, asymmetric scheme published in 1976 by W. Diffie and M. Hellman
  - Based on the difficulty to compute **discrete logarithms**

- The two parties need to agree (also publicly) on two parameters p, g
  - p is a **large prime number**
  - g is a **primitive root modulo p**
    - $\{g^k \bmod p \mid 1 \leq k < p\} = \{1, 2, \ldots, p-1\}$

Protocol Flow



Problems with Specific Private Values

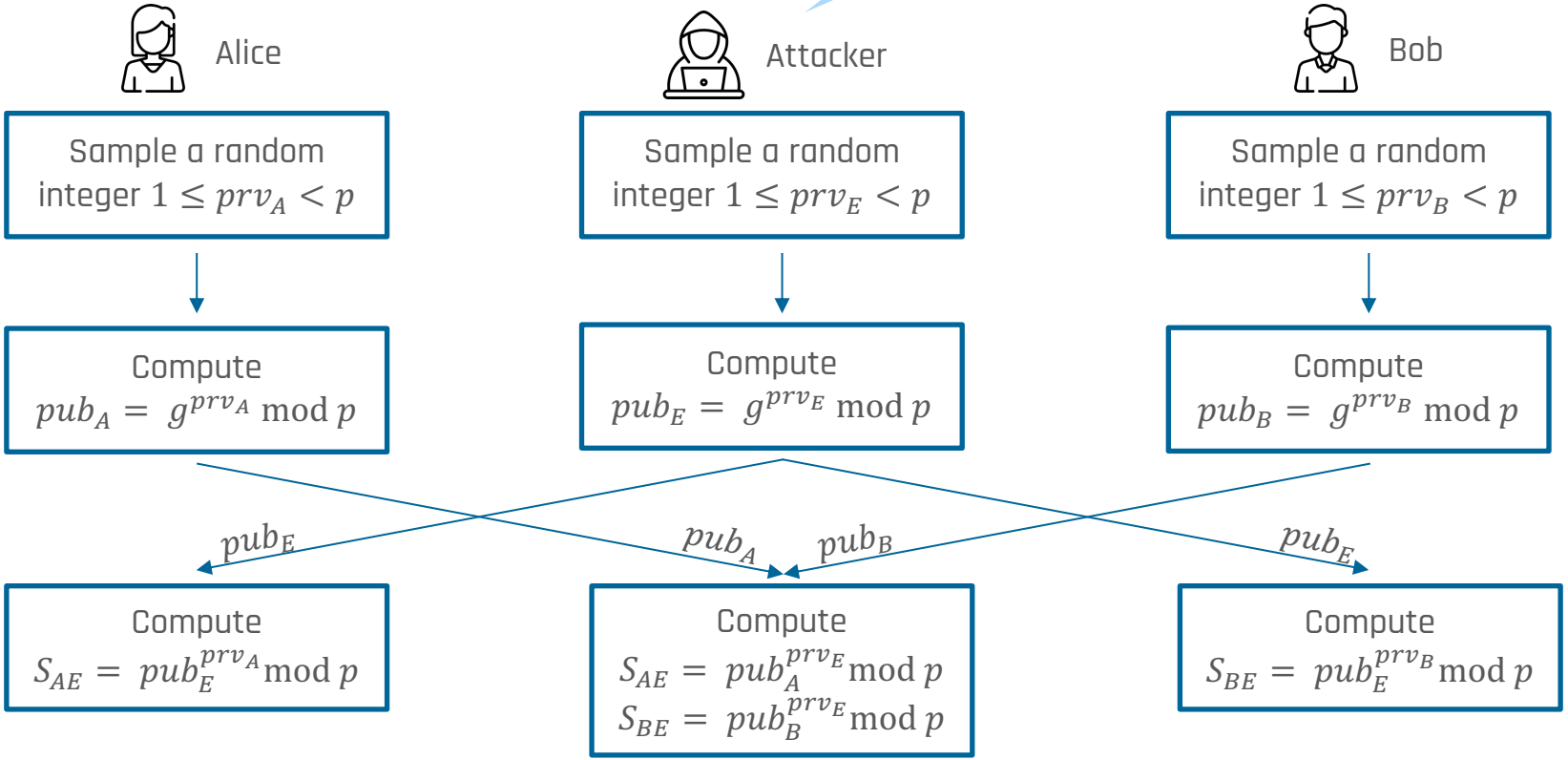
- If Alice chooses $prv_A = 1$, then $pub_A = g$, which is publicly known
  - The shared secret is $S = pub_B$, which is known to an attacker (sent over the public channel)

- If Alice chooses $prv_A = p - 1$, then $pub_A = 1$ (Fermat's little theorem)
  - The shared secret is $S = 1$, irrespective of the value prv_B chosen by Bob

- The private value should be randomly selected from the set $\{2, \dots, p - 2\}$

Man-in-the-Middle Attack

The attacker can **block all messages** between Alice and Bob



Man-in-the-Middle Attack

The attacker can **block all messages** between Alice and Bob

Alice

Attacker

Bob

Sample a random integer $1 \leq prv_A < p$

Sample a random integer $1 \leq prv_E < p$

Sample a random integer $1 \leq prv_B < p$

Compute $pub_A = g^{prv_A} \bmod p$

Compute $pub_E = g^{prv_E} \bmod p$

Compute $g^{prv_B} \bmod p$

pub_E

pub_A

pub_B

b_E

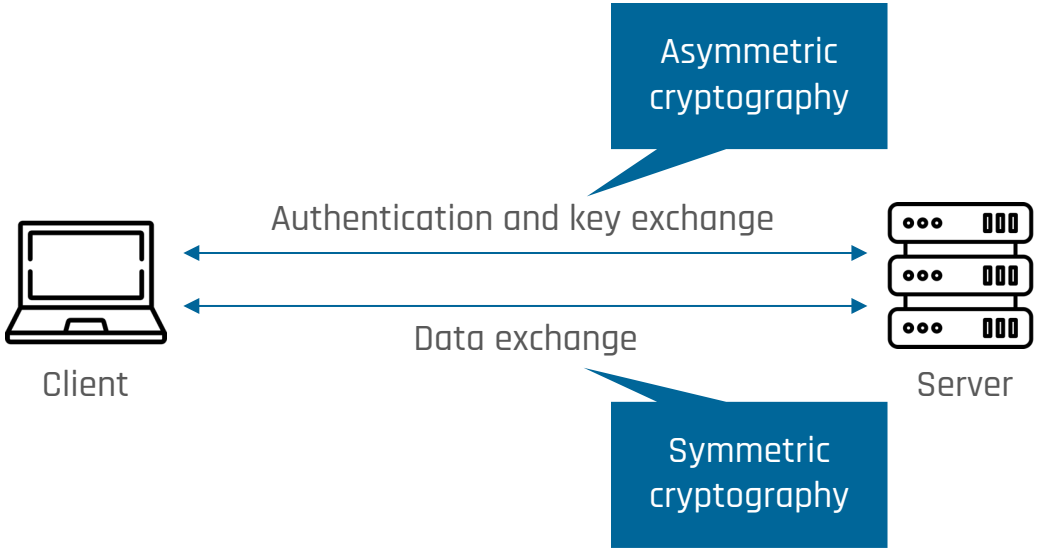
Compute $S_{AE} = pub_E^{prv_A} \bmod p$

Compute $S_{AE} = pub_A^{prv_E} \bmod p$
$S_{BE} = pub_B^{prv_E} \bmod p$

ompute $S_{BE} = pub_E^{prv_B} \bmod p$

Attack due to **missing authentication**!
Solution consists in the usage of **certificates** (next lecture)

Setup of a Connection in Internet



Thank You!