Deterministic algorithm.

- **P** - set of all decision problems that are solvable in polynomial time.
- **NP** - set of all decision problems for which a "yes" can be verified in polynomial time given a certificate (or a witness). NOTE: Certificate may be difficult to obtain.
Deterministic algorithm: NP

Problem $COMP$: Is $n \in \mathbb{N}$ composite. $COMP \in \textbf{NP}$

- Given a divisor of $n$ we can easily verify.
- Divisor may be difficult to obtain.
- Not known whether $COMP \in \textbf{P}$ or not.
\( \mathcal{N} \) is a polynomial-time bounded nondeterministic Turing machine with exactly two nondeterministic choices at each step. 

\( \mathcal{N} \) is called *precise* if all its computations on input \( x \) halt after the same number of steps. 

The output of a probabilistic machine \( \mathcal{N} \) on input \( x \) is a random variable \( M(x) \).

\( P(M(x) = y) \) is the probability that machine \( M \) on input \( x \) outputs \( y \).

The probability space is the space of all possible outcomes of the internal coin flips of \( \mathcal{N} \) taken with uniform probability distribution.
A probabilistic polynomial-time Turing machine $\mathcal{N}$ accepts a language $L$ if the following is true for each input $x$:

1. if $x \in L$ then at least half of computations output “yes”:
\[
P(\mathcal{N}(x) = 1) \geq \frac{1}{2};
\]

2. if $x \notin L$ then all computations output “no”:
\[
P(\mathcal{N}(x) = 0) = 1.
\]
Probabilistic algorithm: class RP

Randomized Polynomial Time (RP)

RP is the class of all languages accepted by probabilistic polynomial-time Turing machines.
Probabilistic algorithm: errors

- **False positive error**: output "yes" when true is "no"
- **False negative error**: output "no" when true is "yes"

RP is a class of problems that have a probabilistic algorithm with no false positives and limited number of false negative errors.
Easy to see that $\mathbf{P} \subset \mathbf{RP} \subset \mathbf{NP}$.

- A polynomial deterministic algorithm is a special case of a probabilistic algorithm and decides $L$ with probability 1 which is greater than $\frac{1}{2}$.

- A probabilistic polynomial Turing machine is a polynomial nondeterministic machine since at least half of computations accept a string in $L$ and, therefore, there is at least one computation that decides $L$. 
**Probabilistic algorithm: ZPP**

**coRP** is a class of problems that have a probabilistic algorithm with no false negatives and limited number of false positive errors.

**Zero Error Probability Polynomial Time (ZPP)**

**ZPP** is the class of languages from

\[ \text{RP} \cap \text{coRP} \]

- By executing two probabilistic algorithms from **RP** and **coRP** on the same input in parallel we guaranteed to obtain a correct answer.
- Algorithms of the type described above are called *Las Vegas* algorithms.
Probabilistic algorithm: example

- $PRIMES \in \text{RP}$

- Actually we know that $PRIMES$ is in $\text{P}$
Probabilistic algorithm: BPP

Definition (Bounded Error Probability Polynomial Time)

A language $L$ is in the class BPP if there is a probabilistic polynomial-time Turing machine $\mathcal{N}$ such that for all inputs $x$:

1. if $x \in L$ then $P(\mathcal{N}(x) = 1) \geq \frac{2}{3}$;
2. if $x \notin L$ then $P(\mathcal{N}(x) = 0) \geq \frac{2}{3}$;

– BPP considered to be class of problems solved by ”practical“ algorithms.
Probabilistic algorithm: BPP

- The constant $\frac{2}{3}$ indicates that the probability of the correct answer is bounded away from half to make the probability amplification computations efficient.
- It can be shown that the definition is robust if we substitute the number $\frac{2}{3}$ by

\[
\frac{1}{2} + \frac{1}{p(|x|)} \quad \text{or} \quad 1 - \frac{1}{2p(|x|)}
\]

where $p$ is a polynomial.
Suppose we have a public-key cryptographic scheme.

- Legitimate parties should be able to decode the secret efficiently, which means that there exist a polynomial-time verifiable witness to the decoding and the problem of breaking a cryptographic scheme is in $\text{NP}$.
- For a cryptographic scheme to be considered secure there should be no practical algorithm to break the encryption.
- Therefore, if a secure cryptographic scheme exists then $\text{NP} \not\subseteq \text{BPP}$. 
Complexity and Cryptography

- Whether BPP contains NP is an open problem.
- NP $\not\subseteq$ BPP is necessary, but not sufficient for a secure cryptographic scheme to exist.
- The condition which bounds away the probability of an error must hold for all inputs. In this sense BPP is analogues to P, i.e. worst case.
The positive answer to the problem $\text{NP} \not\in \text{BPP}$ may have no practical implications for cryptography.

We need problems which belong in $\text{NP} \setminus \text{BPP}$ and are hard on a significantly large fraction of inputs.

A problem may be considered hard if there is no efficient algorithm which solves the problem on any but strongly negligible set of inputs.
Let $P_1$ and $P_2$ be two problems. $P_1$ is said to polytime reduce to $P_2$ if there is an algorithm $A$ that solves $P_1$ which uses, as subroutine, an algorithm $B$ which solves $P_2$ and $A$ runs in polynomial time if $B$ does.

If $L_1$ polytime reduces to $L_2$ then $L_2$ is at least as hard as $L_1$. 
Suppose $P$ is a "hard" theoretical problem and $P_C$ is a problem of cryptanalysis of a cryptographic scheme $C$.

If $P$ polytime reduces to $P_C$ then breaking $C$ is at least as hard as solving $P$.

In other words if we can break $C$ then we can solve $P$.

Finding hard problems is of great practical importance.
Problem: Since we do not know whether $P = NP$ or not, there are no provably hard problems. Based on experience certain problems believed to be hard:

- Integer factorization
- Discrete algorithm
- Linear codes
- Elliptic curves
- Systems of multivariate equations.
Let $P_1$ and $P_2$ be two problems. $P_1$ is said to polytime reduce to $P_2$, denote $P_1 \leq_P P_2$, if there is an algorithm $A$ that solves $P_1$ which uses, as subroutine, an algorithm $B$ which solves $P_2$ and $A$ runs in polynomial time if $B$ does.

Denote $P_1 \leq_{RP} P_2$ if algorithm $B$ is probabilistic.

If $P_1 \leq P_2$ and $P_2 \leq P_1$ then two problems are computationally equivalent.
RSA Public-key Cryptosystem (Rivest, Shamir, Adleman)

Recall

- \( \mathbb{Z}_n^\times = \{ a \in \mathbb{Z}_n \mid \gcd(a, n) = 1 \} \)
- \( \mathbb{Z}_n^\times \) is a finite abelian group.
- \( |\mathbb{Z}_n^\times| = \varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right) \) (Euler’s Totient function)
- If \( n = pq \), then \( \varphi(n) = (p - 1)(q - 1) \)
- \( a^{\varphi(n)} = 1 \mod n \) for all \( a \in \mathbb{Z}_n^\times \)
RSA: key generation

RSA Public-Key Cryptosystem

Key Generation

Alice performs the following steps:

1. Generate two distinct random primes $p$ and $q$ such that $\log p \approx \log q$ (about the same size)
2. Compute $n = pq$ and $\varphi = (p-1)(q-1)$
3. Select $e \in (1, \varphi)$ at random and such that $\gcd(e, \varphi) = 1$
4. Compute $d$ - the inverse of $e$ modulo $\varphi$, i.e. $ed = 1 \mod \varphi$

- Alice’s public key: $(n, e)$
- Alice’s private key: $d$
RSA: encryption

Encryption
Bob encrypts message $m$ using Alice’s public key $(n, e)$
1. Encode the message as an integer $m \in [0, n - 1]$
2. Compute $c = m^e \mod n$
3. Send the ciphertext $c$ to Alice

Decryption
Alice recovers $m$ from $c$:
1. $m = c^d \mod n$
**RSA: encryption**

*Proof:*

Show that \( m = c^d \mod n \), where \( c = m^e \mod n \) and \((n, e), d\) are the public/private keys

1. \( c^d \mod n = m^{ed} \mod n \)
2. \( ed = 1 \mod \varphi(n) \Rightarrow ed - 1 = k\varphi(n) \)
3. \( m^{ed-1} = m^{k\varphi(n)} = 1 \mod n \)
4. Therefore, \( m^{ed-1} = 1 \mod n \) and \( m^{ed} = m \mod n \)
The problem of computing RSA secret key (exponent $d$) given public key $(n, e)$ is computationally equivalent to the problem of factoring $n$.

Obviously if we can factor $n = pq$ then we know $\varphi = (p - 1)(q - 1)$ and $d$ is obtained by computing inverse of $e$ modulo $\varphi$. 
There is an efficient probabilistic algorithm which computes factorization of $n$ given the secret key (exponent $d$) and the public key ($n, e$).

**Proof:**

- $ed - 1 = k\varphi(n) \mod n$
- Therefore, $a^{ed-1} = 1 \mod n$, for all $a \in \mathbb{Z}_n^\times$
- Suppose, $ed - 1 = 2^s t$, and $t$ is odd.
- **Fact:** There exists $i \in [1, s]$ s.t.
  \[
  a^{2^i t - 1} \neq \pm 1 \mod n \text{ and } a^{2^i t} = 1 \mod n
  \]
  for at least half of all $a \in \mathbb{Z}_n^\times$
- For such $i$ $\gcd(a^{2^i-1} t, n)$ is nontrivial factor of $n$
Factoring algorithm when $ed - 1$ is known.

1. Select $a \in \mathbb{Z}_n^\times$ at random;
2. Check if $i \in [1, s]$ satisfying condition above exists

- The expected number of trials is 2.
The RSA Problem (RSAP)

Given a positive integer \( n = pq \), where \( p, q \) are distinct odd primes, a positive integer \( e \) such that \( \gcd(e, (p - 1)(q - 1)) = 1 \) and an integer \( c \), find an integer \( m \) such that

\[
m^e = c \mod n.
\]

- Security of the RSA encryption is based on RSAP.
- \( RSAP \leq P \) FACTORING
- \( RSAP \) is believed to be equivalent to FACTORING but no proof.
RSA: small encryption exponent.

- For a random $e$ the expected time to encrypt is $k$ squarings plus $k/2$ multiplications, where $k$ is a bit length
- if $e$ is small or has few ones then time can be reduced
- Popular choices: $e = 3$, $e = 2^{16} + 1 = 10000000000000001$
Suppose $e = 3$ and suppose that the same message is sent to three different agents.

We have a system of equations:

$$\begin{cases}
x = m^3 \mod n_1 \\
x = m^3 \mod n_2 \\
x = m^3 \mod n_3
\end{cases}$$
RSA: small encryption exponent.

- Most likely $n_1, n_2, n_3$ will be pairwise relatively prime.
- We can use Chinese Remainder Theorem
- The system can be solved for $x$ using Gauss’s algorithm
- Since $m^3 < n_1 n_2 n_3$ then by CRT $x = m^3$
RSA: small encryption exponent.

- Small exponent is a problem when message $m < n^{1/e}$. Enough to compute $e$th integer root.
- One can append a pseudorandom string to the message before encryption - *salting*
RSA: Coppersmith attack.

Attacks exist

- When $d$ is small.
- If half if the higher bits of $p$ is known. (Coppersmith)
- ...
RSA: parameters.

- $p$ and $q$ of about the same length and sufficiently large
  $n \geq 1024$ (elliptic curve factoring)
- Modulus $n$ should be different
- Difference $p - q$ should not be too small. Otherwise just try
  all odd integers around $\sqrt{n}$
- Strong primes (not completely clear how much security
  gained).
- Take care of small exponents.
RSA: parameters.

**Strong primes**

A prime $p$ is called a *strong prime* if there exist $r, s, t$ such that

- $p - 1$ has a large prime factor $r$;
- $p + 1$ has a large prime factor $s$;
- $r - 1$ has a large prime factor $t$. 
**Gordon’s algorithm**

**INPUT:** \( n \) - bit length of the prime

**OUTPUT:** A strong prime of bit length \( n \)

1: Generate two primes \( s \) and \( t \) of size roughly equal to \( n/2 \)

2: Select integer \( i_0 \). Find the first prime
   \[ r = 2it + 1, \; i = i_0, i_0 + 1, i_0 + 2, \ldots \]

3: Compute \( p_0 = (2s^{r-2} \mod r)s - 1 \)

4: Select integer \( j_0 \). Find the first prime
   \[ p = p_0 + 2jrs, \; j = j_0, j_0 + 1, j_0 + 2, \ldots \]

5: **RETURN** \( p \)
RSA: parameters.

Proof.

- Assume \( r \neq s \) then \( s^{r-1} = q \mod r \)
- \( p_0 = (2s^{r-2} \mod r)s - 1 = 1 \mod r \)
- \( p_0 = -1 \mod s \)
- \( p - 1 = p_0 + 2jrs - 1 = 0 \mod r \rightarrow r \mid p - 1 \)
- \( p + 1 = p_0 + 2jrs + 1 = 0 \mod s \rightarrow s \mid p + 1 \)
- \( r - 1 = 2it = 0 \mod t \rightarrow t \mid r - 1 \)
RSA: parameters.

- Need to choose sizes of primes $s$, $t$ and integers $i_0$, $j_0$
- Length of $s$, $r$ is roughly half of the bit length of $p$
- Length of $t$ is slightly less than length of $r$
Rabin: key generation

Key Generation
Alice performs the following steps:

1. Generate two distinct random primes \( p \) and \( q \) such that \( \log p \approx \log q \)
2. Compute \( n = pq \)

- Alice’s public key: \( n \)
- Alice’s private key: \( (p, q) \)
## Rabin: encryption

### Encryption

Bob encrypts message $m$ using Alice's public key $n$

1. Encode the message as an integer $m \in [0, n - 1]$
2. Compute $c = m^2 \mod n$
3. Send the ciphertext $c$ to Alice

### Decryption

Alice recovers $m$ from $c$:

1. Compute the four square roots $m_1, m_2, m_3, m_4$ of $c$ modulo $n$
2. Somehow choose which of the four is the true message $m$. 
Security of Rabin encryption is based on SQROOT:

Square Root Problem (SQROOT)

Given a composite integer $n$ and a quadratic residue $a$ modulo $n$, find a square root of $a$ modulo $n$. 
SQROOR is NOT a special case of RSAP

Proof.

- Recall $\gcd(e, (p - 1)(q - 1)) = 1$
- Since $(p - 1)(q - 1)$ is even then $e$ must be odd
- Therefore $e \neq 2$
FACTORIZATION $\leq_{RP} SQROOT$.

Proof.

- Suppose there exists a polynomial time algorithm $A$ which solves $SQROOT$
- Select $x$ randomly such that $\gcd(x, n) = 1$ and compute $a = x^2 \mod n$
- Execute $A$ to compute square root $y = a^2 \mod n$
- **FACT:** if $x^2 = y^2 \mod n$ and $x \neq \pm y \mod n$ then $\gcd(x - y, n)$ is a nontrivial factor of $n$
- If $y = \pm x \mod n$ then FAIL
- If $y \neq \pm x \mod n$ then $\gcd(x - y, n)$ is a nontrivial factor of $n$
- The expected number of trials is 2 (four square roots total)
Rabin encryption algorithm is the first encryption provably secure against a passive adversary.

Equivalence of $SQROOT$ and $FACTORING$ can be used chosen-ciphertext attack:

- Adversary selects message $m$ at random and sends $c = m^2 \mod n$ to Alice
- Alice returns plaintext $y$
- Since $m$ random, probability that $y \neq \pm m \mod n$ is 0.5
- And $gcd(m - y, n)$ is one of the factors of $n$
Rabin: redundancy

Adding pre-specified redundancy to the message

- Helps to choose the true plaintext.
- Helps avoid chosen-ciphertext attack
The discrete logarithm problem

Discrete logarithm

Let $G$ be a finite cyclic group of order $n$ and $a$ be a generator of $G$. Let $b \in G$. The \textit{discrete logarithm of $b$ to the base $a$}, denoted $\log_a b$, is the unique integer $x$ such that $0 \leq x \leq n - 1$, such that $a^x = b$.

- Example: $\mathbb{Z}_{97}^\times = < 5 >$.
  
  \begin{itemize}
  \item Since $5^{32} = 35 \mod 97$,
  \item $\log_5 35 = 32$ in $\mathbb{Z}_{97}^\times$
  \end{itemize}
The discrete logarithm problem

Discrete logarithm problem

Given a prime $p$, a generator $a$ of $\mathbb{Z}_p^\times$ and an element $b \in \mathbb{Z}_p^\times$, find an integer $x$, $0 \leq x \leq p - 2$ such that

$$a^x = b \mod p.$$ 

The Generalized Discrete Logarithm Problem is defined for any finite cyclic group $G$. 
The discrete log: properties

- \( \log_a(bc) = \log_a b + \log_a c \mod n \)
- \( \log_a(b^c) = c \log_a b \mod n \)
- Difficulty of DLP does not depend on generator:
  - Let \( G = \langle \alpha \rangle = \langle \beta \rangle \), \( |G| = n \)
  - Let \( x = \log_\alpha b, y = \log_\beta b \) and \( z = \log_\alpha \beta \)
  - \( \alpha^x = b = \beta^y = (\alpha^z)^y \). Therefore \( x = zy \mod n \)
  - \( \log_\beta b = (\log_\alpha b)(\log_\alpha \beta)^{-1} \mod n \)
  - Any algorithm that computes to the base \( \alpha \) can be used to compute to the base \( \beta \)
The discrete log: algorithms

- Baby-step giant-step: $O(\sqrt{n})$
- Pollard’s $\rho$ algorithm: expected time $O(\sqrt{n})$ but negligible storage.
- Pohlig-Hellman: $O(\sum_{i=1}^{r} e_i (\lg n + \sqrt{p_i}))$, where $n = p_1^{e_1} p_2^{e_2} \ldots p_r^{e_r}$
- Index-calculus algorithms.
The discrete log: algorithms

Deffie-Hellman key exchange

Alice performs the following steps:

1. Publish a large prime $p$ and generator $\alpha$ of $\mathbb{Z}_p^\times$ (One time step)
2. Alice chooses a random secret $a \in [1, p - 2]$ and sends $\alpha^a \mod p$ to Bob
3. Bob chooses a random secret $b \in [1, p - 2]$ and sends $\alpha^b \mod p$ to Alice
4. Alice chooses the shared key $K = (\alpha^b)^a \mod p$.
5. Bob chooses the shared key $K = (\alpha^a)^b \mod p$.

- Secure against passive eavesdropper
- No authentication - not from active adversary (man in the middle attack)
The discrete log: algorithms

Key Generation

Alice performs the following steps:

1. Generate a large prime \( p \) and generator \( \alpha \) of \( \mathbb{Z}_p^\times \)
2. Choose random \( a \in [1, p-2] \) and compute \( \alpha^a \mod p \)

- Alice’s public key \((p, \alpha, \alpha^a)\)
- Alice’s private key \(a\)
The discrete log: algorithms

### Encryption

Bob encrypts message $m$ using Alice’s public key $(p, \alpha, \alpha^a)$

1. Encode the message as an integer $m \in [0, p - 1]$
2. Select random $b \in [1, p - 2]$
3. Compute $\beta = \alpha^b \mod p$ and $\delta = m \cdot (\alpha^a)^b \mod p$
4. Send the ciphertext $c = (\beta, \delta)$ to Alice

### Decryption

Alice recovers $m$ from $c$:

1. Recover $m = \beta^{-a} \delta \mod p$. 
The Diffie-Hellman problem

Diffie-Hellman problem (DF)

Given a prime $p$, a generator $\alpha$ of $\mathbb{Z}_p^\times$ and elements $\alpha^a \mod p$ and $\alpha^b \mod p$, find

$$\alpha^{ab} \mod p.$$ 

Generalized DF problem is defined for arbitrary finite cyclic group $G$
The Diffie-Hellman problem

\[ \text{DHP} \leq_P \text{DLP} \quad (\text{GDHP} \leq_P \text{GDLP}) \]

Not known if equivalent. Some results for smooth primes.
The Diffie-Hellman: finding a generator

Generator of cyclic group

**INPUT:** \( G, \; |G| = n = p_1^{e_1} \cdots p_k^{e_k} \)

**OUTPUT:** \( \alpha \) s.t. \( G = \langle \alpha \rangle \)

1: Choose random \( \alpha \in G \)

2: FOR \( i = 1, \ldots, k \)

3: Compute \( b = \alpha^{n/p_i} \)

4: IF \( b = 1 \) then GOTO 1

5: RETURN \( \alpha \)
The Diffie-Hellman: finding a generator

- Correctness from the fact that the order of an element must divide order of the group.
- \( G \) has exactly \( \varphi(n) \) generators
- Probability of a random element being a generator

\[
\frac{\varphi(n)}{n} \geq \frac{1}{6 \ln \ln n}
\]
Safe prime is a prime of a form $p = 2q + 1$, where $q$ is a prime.
The Diffie-Hellman: selecting $p$ and $\alpha$

Selecting safe prime and a generator of a cyclic group

**INPUT:** Bit length of a prime $k$

**OUTPUT:** a $k$-bit safe prime $p$ and $\alpha$ s.t. $\mathbb{Z}_p^\times = \langle \alpha \rangle$

1: DO

2: Select random $(k - 1)$-bit prime $q$

3: Compute $p = 2q + 1$

4: UNTIL $p$ is prime

5: Find generator $\alpha$ with $n = 2q$

6: RETURN $(p, \alpha)$
RSA is a modular equation.

Natural extension to have several modular equations over several variables.
Let $\mathbb{F}_q$, $q = p^m$ be a field.

Let

$$f(a) = \sum_{i,j} \beta_{ij} x^{q^i j} + \sum k \alpha_k x^{q^k \mu} \in \mathbb{F}_q^n[a]$$

Recall that extension

$$\mathbb{F}_{q^n} \simeq \mathbb{F}_q[x]/g(x), g(x) \in \mathbb{F}_q[x]$$

is irreducible of degree $n$

Therefore elements of $\mathbb{F}_{q^n}$ can be represented as $n$-tuples over $\mathbb{F}_q$ and every function $f: \mathbb{F}_q^n \to \mathbb{F}_q^n$ can be represented as a single polynomial as well as $n$ polynomials in $n$ variables.

$$f(a_1, \ldots, a_n) = (f_1(a_1, \ldots, a_n), \ldots, f_n(a_1, \ldots, a_n))$$

$f_i$ are quadratic
A transformation \( t : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^n \) is affine if every coordinate is transformed by a linear transformation plus translation.

It is easy to compute \( t^{-1} \) if \( t \) is affine.
Hidden Fields Equations.

Let $F_q$ be field. (Typically $\mathbb{F}_2$)

**Key Generation**

- Choose secret function $f$ and two secret affine bijections $s$, $t$
  $$F_q^n \rightarrow F_q^n$$

- Compute public polynomials $P_1, \ldots, P_n$ of $n$ variables such that
  $$t(f(s(x_1, \ldots, x_n))) = (P_1(x_1, \ldots, x_n), \ldots, P_n(x_1, \ldots, x_n))$$
Hidden Fields Equations.

Encryption
- Represent a message $m$ as an $n$-tuple $(m_1, \ldots, m_n)$ over $\mathbb{F}_q$
- Compute ciphertext
  \[ c = (c_1, \ldots, c_n) = (P_1(m_1, \ldots, m_n), \ldots, P_n(m_1, \ldots, m_n)) \]

Decryption
- Compute all solutions $z$ to equation
  \[ f(z) = t^{-1}(c) \]

   NOTE: $f$ is the polynomial in one variable.
- Compute $s^{-1}(z)$, for each solution $z$.
- Choose the original message $m$. 
Hidden Fields Equations.

- To break HFE one has to solve a system of $n$ quadratic equation over $n$ variables
- $NP$-hard in general
- Seems to be hard when parameters are chosen carefully
Commutator Key Exchange: key generation.

Let $G$ be a group (could be non-commutative and infinite).

- Alice randomly generates an $n$-tuple of elements from $G$
  \[ \bar{a} = \{a_1, \ldots a_n\} \]
- Bob randomly generates an $m$-tuple of elements from $G$
  \[ \bar{b} = \{b_1, \ldots b_m\} \]
- Alice randomly generates a product
  \[ A = a_{s_1}^{\varepsilon_1} \cdots a_{s_k}^{\varepsilon_k}, \varepsilon_i = \pm 1 \]
- Bob randomly generates a product
  \[ B = b_{t_1}^{\delta_1} \cdots b_{t_k}^{\delta_k}, \delta_i = \pm 1 \]

- $\bar{a}, \bar{b}$ are public sets
- Elements $A, B$ are private keys.
Commutator Key Exchange: shared key generation

- Alice computes
  \[ b'_i = A^{-1} b_i A, \ 1 \leq i \leq m \]
  and sends them to Bob.

- Bob computes
  \[ a'_i = B^{-1} a_i B, \ 1 \leq i \leq n \]
  and sends them to Alice.

- Alice computes
  \[ K_A = A^{-1} a'_1 \ldots a'_{s_k} = A^{-1} B^{-1} AB \]

- Bob computes
  \[ K_B = b'_1 \ldots b'_1 B = A^{-1} B^{-1} AB \]

Thus, Alice and Bob obtain the same shared key
\[ K = K_A = K_B = A^{-1} B^{-1} AB \]
To break one needs to solve the following system of equations:

\[
\begin{align*}
    a'_1 &= X^{-1}a_iX \\
    a'_2 &= X^{-1}a_2X \\
    &\quad \ldots \\
    a'_n &= X^{-1}a_nX
\end{align*}
\]
Commutator Key Exchange.

Conjugacy Problem
Given a group $G$ and two elements $a, b \in G$ decide whether there exists an element $x$ such that

$$x^{-1}ax = b.$$ 

One of the fundamental problems in group theory.

Undecidable in general.
The problem of solving the system of conjugacy equations is called simultaneous conjugacy search problem.

The security is based on the hardness of this problem.
The Subset Sum problem

Decision

Given $a_1, \ldots, a_n, s \in \mathbb{N}$, are there $x_1, \ldots, x_n \in \{0, 1\}$ such that

$$\sum_{i=1}^{n} a_i x_i = s.$$ 

– Search problem requires to find $x_1, \ldots, x_n$
The Subset Sum problem

- Solving Subset Sum is \textit{NP}-complete (search is \textit{NP}-hard)
- It is possible to have a trapdoor i.e. decoding is easy when secret information is known.
The Subset Sum problem

- The sequence $b_1, b_2, \ldots, b_n$ is called superincreasing if

\[ b_i > \sum_{j=1}^{i} b_j \]

- It can be shown that Subset Sum problem for superincreasing sequences is easy.
- There is a unique solution for $x_1, \ldots, x_n$. 
Merkle-Hellman Knapsack

Key generation

1. Fix \( n \). Choose superincreasing sequence \((b_1, \ldots, b_n)\) and modulus \( M > b_1 + \ldots + b_n \)
2. Select random integer \( W \in [1, M - 1] \), such that \( \gcd(W, M) = 1 \)
3. Select random permutation \( \pi \) of \( \{1, \ldots, n\} \)
4. Compute \( a_i = Wb_{\pi(i)} \mod M \), \( i = 1, 2, \ldots, n \)

- Alice’s public key: \((a_1, \ldots, a_n)\)
- Alice’s private key: \((\pi, M, W, (b_1, \ldots, b_n))\)
# Merkle-Hellman Knapsack

## Encryption

1. Represent message $m$ as a binary string $m = m_1 m_2 \ldots m_n$, $m_i \in \{0, 1\}$
2. Compute ciphertext

\[ c = m_1 a_1 + m_2 a_2 + \ldots + m_n a_n \]

## Decryption

1. Compute $d = W^{-1} c \mod M$
2. Find $r_1, r_2, \ldots, r_n, r_i \in \{0, 1\}$ such that

\[ d = r_1 b_1 + r_2 b_2 + \ldots r_n b_n \]
3. Recover $m$, where bits $m_i = r_{\pi(i)}$
Merkle-Hellman Knapsack

\[ d = W^{-1} c = W^{-1} \sum_{i=1}^{n} m_i a_i = \sum_{i=1}^{n} m_i b_{\pi(i)} \mod M \]

Since \( \leq 1 < M \), \( r_i \)'s give the message bits after permutation.
The Subset Sum problem

– In general NP complete

Density of knapsack

Let $S = \{a_1, \ldots, a_n\}$ be a knapsack set. the density of $S$

$$d = \frac{n}{\max \{\log a_i | 1 \leq i \leq n\}}$$

– Low density can be solved by LLL which finds the shortest vector
The Subset Sum problem

Consider Lattice $L \in \mathbb{Z}^{n+1}$ generated by the rows of the matrix

$$
\begin{pmatrix}
1 & 0 & \cdots & 0 & -a_1 \\
0 & 1 & \cdots & 0 & -a_2 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & 1 & -a_n \\
0 & 0 & \cdots & 0 & s
\end{pmatrix}
$$
The Subset Sum problem

- Let \( r_1, \ldots, r_{n+1} \) be the basis of \( L \) (columns of the matrix)
- Let \( (x_1, \ldots, x_n) \in \{0, 1\}^n \) be a solution to the Subset Sum problem
- Then
  \[
  v = \sum_{i=1}^{n} x_i r_i + r_{n+1} = (x_1, \ldots, x_n, 0) \in L
  \]
- Note \( \|v\|_2 \leq \sqrt{n} \) which is small since \( a_i \) are large.
- Often this is the shortest vector found by LLL algorithm.
- Works well for low-density subset problems. For Merkle & Hellman density is \( 1/n \)
The Subset Sum problem

- Merkle-Hellman and its variants are broken
- Systems which do not use modular multiplication to hide ease subset sum problems seem more promising (Chor-Rivest scheme)