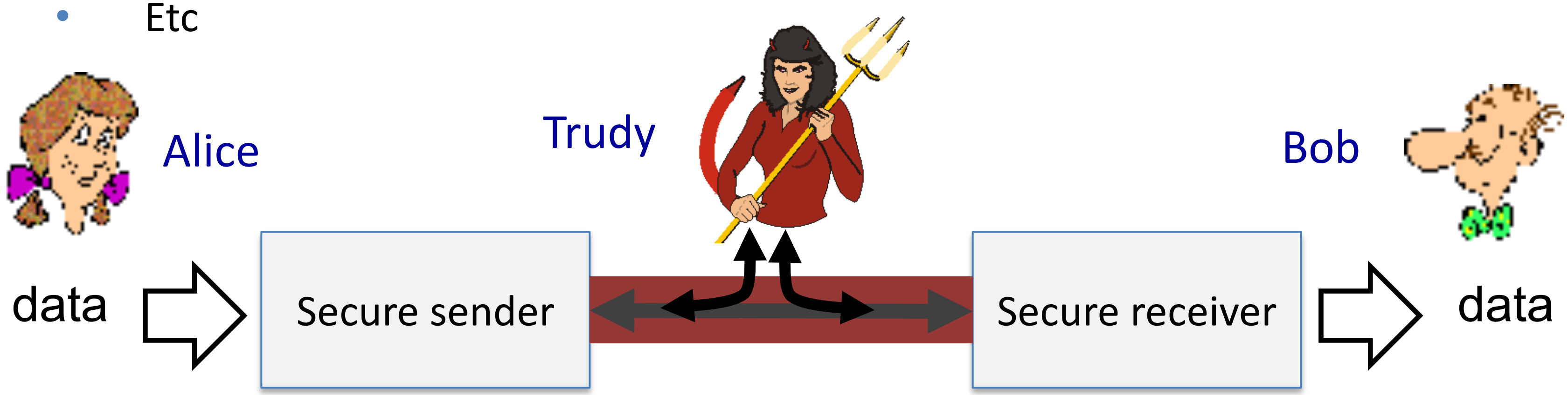
Cryptographic fundamentals, Asymmetric cryptography and Homomorphic encryption

Artem Lenskiy

(Artem.Lenskiy@anu.edu.au)

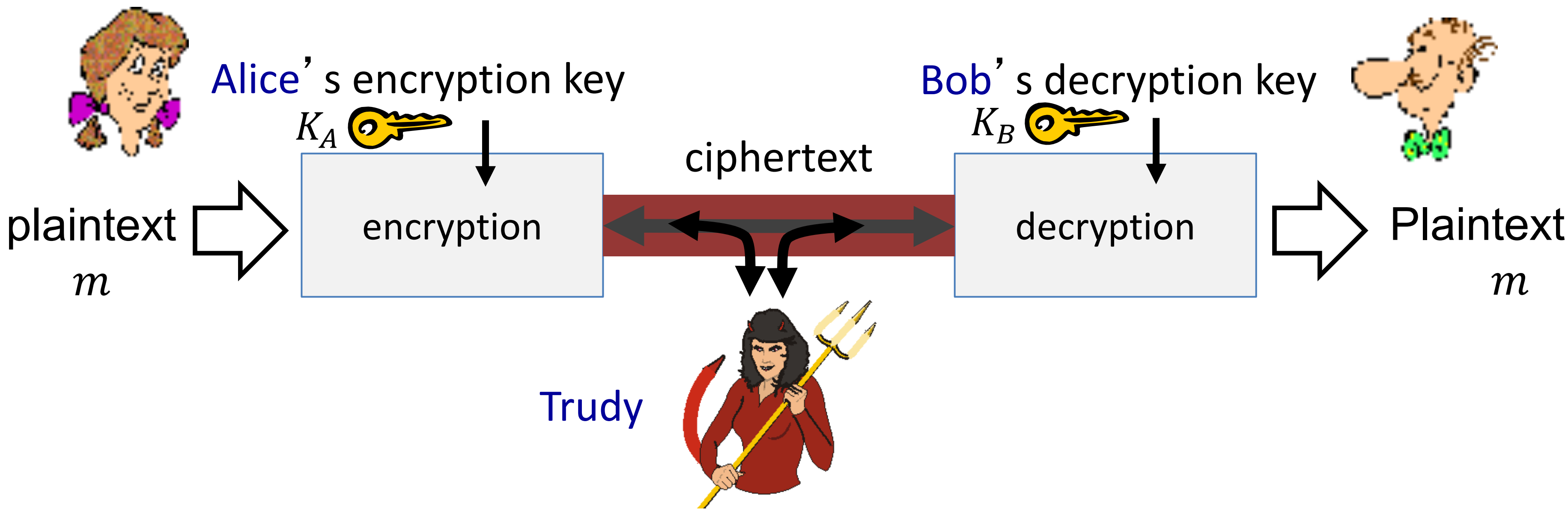
Information security

- The practice of protecting information by mitigating information risks.
  - Secure storage
  - Secure communication
  - Secure computation
  - Etc



- Information security uses *cryptography* to transform usable information into a form that renders it unusable by anyone other than an authorized user (encryption).

The language of cryptography



m plaintext message

$K_A(m)$ ciphertext, encrypted with key K_A

$m = K_B(K_A(m))$

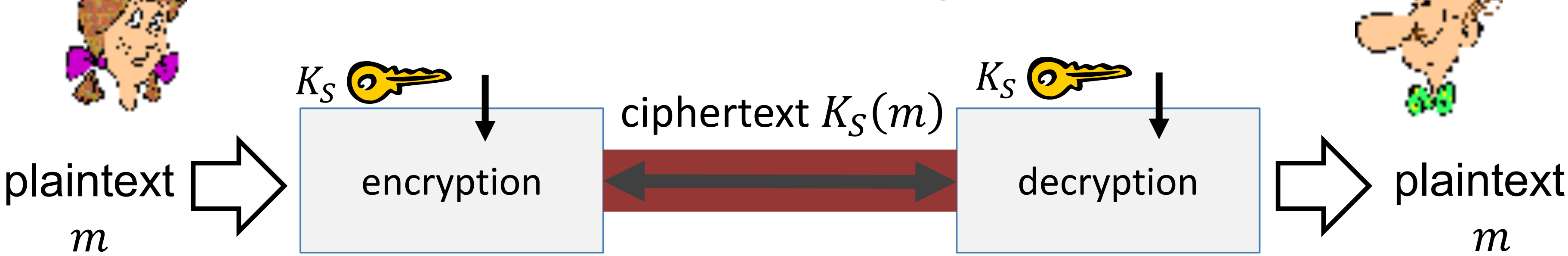
Applications of asymmetric cryptography

- What can go wrong ?
  - *eavesdrop:* intercept messages
  - actively *insert* messages into connection
  - *impersonation:* can fake (spoof) source address in packet (or any field in packet)
  - *hijacking:* "take over" ongoing connection by removing sender or receiver, inserting himself in place
  - *denial of service*: prevent service from being used by others (e.g., by overloading resources)

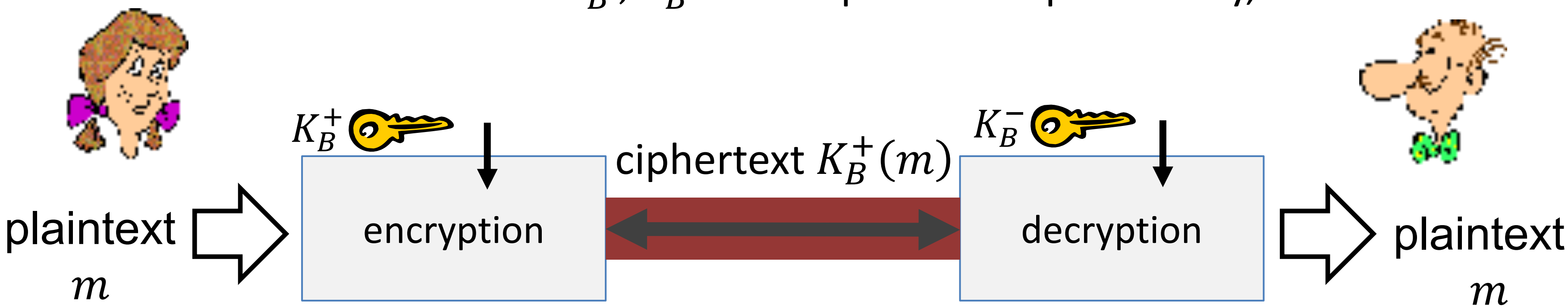
Symmetric and Asymmetric cryptography

- **Symmetric key crypto**: Bob and Alice share same (symmetric) key:

$$K_A = K_B = K_S$$

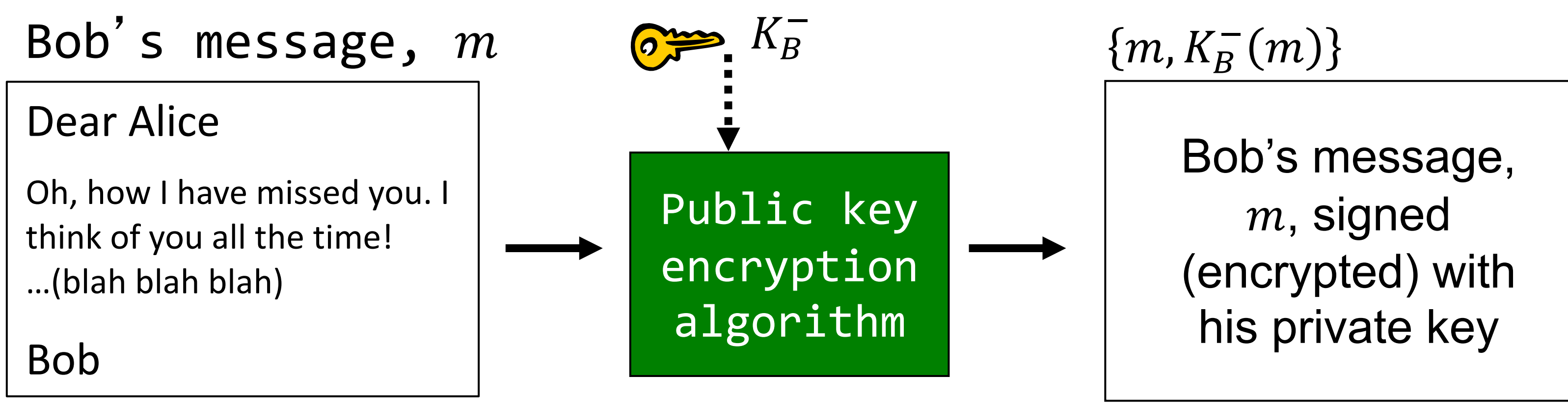


- **Asymmetric key crypto**: K_A^+, K_A^-: Alice's public and private key,
  K_B^+, K_B^-: Bob's public and private key,



Simple digital signature for message m

- **Goal**: sender (Bob) digitally signs document, establishing he is document owner/creator.
- *verifiable, nonforgeable*: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document
- Bob signs m by encrypting with his private key K_B^-, creating "signed" message, $K_B^-(m)$



Simple digital signature for message m

- Suppose Alice receives msg m, with signature: $\{m, K_B^-(m)\}$
- Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$ then checks $K_B^+\left(K_B^-(m)\right) = m$.
- If $K_B^+\left(K_B^-(m)\right) = m$, whoever signed m must have used Bob's private key.

- **Result:**
- Alice thus verifies that
  - Bob signed m
  - no one else signed m
  - Bob signed m and not m'
- **Nonrepudiation:**
  - Alice can take m, and signature $K_B(m)$ to court and prove that Bob signed m.

RSA (Rivest–Shamir–Adleman) algorithm

- ✓ One of the first public-key cryptosystems and is widely used.
- ✓ *Idea*: finding the factors of a large composite number is difficult.

Example: What are the factors of 1027 ? Can you check 13 and 19 ? P vs NP

- ✓ **Modular arithmetic:**

$$x \bmod n = \text{remainder of } x \text{ when divided by } n$$

Properties:

$$[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$$
$$[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$$
$$[(a \bmod n) \cdot (b \bmod n)] \bmod n = (a \cdot b) \bmod n$$

Then

$$[(a \bmod n)^d] \bmod n = [(a \bmod n) \cdot \ldots \cdot (a \bmod n)] \bmod n = a^d \bmod n$$

Example: $x = 14, n = 10, d = 2$

a. $x^d \bmod n \Longrightarrow 14^2 \bmod 10 = 6$

b. $[(x \bmod n)^d] \bmod n \Longrightarrow [(14 \bmod 10)^2] \bmod 10 = 16 \bmod 10 = 6$

Exercise

- Compute
  - $31 \bmod 7$
  - $27 \bmod 7$
  - $(31 + 27) \bmod 7$
  - $(31 \bmod 7 + 27 \bmod 7) \bmod 7$
  - $(31 \cdot 27) \bmod 7$
  - $(31 \bmod 7) \cdot (27 \bmod 7) \bmod 7$
  - $31^3 \bmod 7$
  - $(31 \bmod 7)^3 \bmod 7$

Greatest common divisor

- For two integers x, y, the greatest common divisor of x and y is denoted $\gcd(x, y)$.
- Two nonzero integers a and b is the greatest positive integer d such that d is a divisor of both a and b.
- Examples:
  - $a = 27, b = 21$ then $d = 7$.
    - Divisors of a are $1,3,\mathbf{7},27$
    - Divisors of b are $1,\mathbf{7},21$
  - $a = 24, b = 54$ then $d = 6$.
    - Divisors of a are $1,2,4,\mathbf{6},24$
    - Divisors of b are $1,2,3,\mathbf{6},9,18,27,27,54$
  - $a = 57, b = 63$ then $d = ?$

RSA algorithm

Key generation:

1. Choose two large prime numbers p, q
2. Compute
   1. $n = pq$
   2. $\phi(n) = \phi(p, q) = \phi(p)\phi(q) = (p-1)(q-1)$

   where $\phi(p) = p - 1$. *Note* $\phi(n) = \phi(p, q)$ is coprime with pq.
3. Choose $e \in \big(1, \phi(n)\big)$ coprime with $\phi(n)$
4. Choose d s.t. $(ed - 1) \bmod \phi(n) \equiv 0$

Then (e, n) and (d, n) are the keys.

Example:

1. $p = 13, q = 17$
2. Compute
   1. $n = 221$
   2. $\phi(n) = (13 - 1)(17 - 1) = 192$
3. $e = 11$, it is coprime with $\phi = 192$
4. $d = 35 \Longrightarrow (11 \cdot 35 - 1) \bmod 192 = 0$

The keys are $(11, 221), (35{,}221)$.

$\phi(n)$ is Euler's Totient Function. See proofs of different interesting properties.

RSA: key generation (Matlab)

Part 1: Key generation

```
% (1) select two distinct prime numbers
p = nthprime(1000); q = nthprime(1001);
% (2) compute n and phi(n) that produces a number that is relatively prime to n
n = q * p;
phi = @(p, q) (p - 1) * (q - 1);
% (3) Choose any number 1 < e < phi(n) that is coprime to phi(n);
e = 0;
while(gcd(e, phi(p,q)) ~= 1) % This number is not a divisor of phi(n)
    e = ceil(rand(1) * phi(p,q) + 1); % Randomly peak until the condition is true
end
% (4) Compute d, such that d and e have the same remainder of division by phi.
d = 2;
while(powmod(d*e, 1, phi(p, q)) ~= 1)
    d = d + 1;
end
```

RSA

Encryption/decryption:

1. Divide a message into bit strings s.t. each string corresponds to a decimal number $m < n$.

2. Encrypt: $c = K_e(m)$

   $$c = (m^e) \bmod n$$

3. Decrypt: $m = K_d(c)$

   $$m = \left(c^d\right) \bmod n$$

Example:

1. Since $n = 221$, 7 bits ($127 < 221$) segments suffice.

   Plaintext Hi!

   $$[100100011010010100001]$$

   72 105 33

2. Encrypt $'!'$:

   $$(33^{11}) \bmod 221 = 67$$

Cyphertext $67 \Longrightarrow 'C'$

3. Decrypt:

   $$(67^{35}) \bmod 221 = 33 \Longrightarrow '!'$$

RSA (MATLAB)

```
m   = int64('!');
e   = 11;
d   = 35;
n   = 221;
phi = 192;


c   = mod(m^d, n) % => 59
m   = mod(c^e, n) % => 59
```

```
m   = sym(int64('!'));
e   = sym(11);
d   = sym(35);
n   = sym(221);
phi = sym(192);


c   = mod(m^d, n) % => 67 or ‘C’
m   = mod(c^e, n) % => 33 or 'C’
```

`m^d` causes the overflow

To avoid the overflow, symbolic math is used.

RSA algorithm

Encryption/decryption:

1. Divide a message into bit strings s.t. each string corresponds to a decimal number $m < n$.

2. Encrypt: $c = K_e(m)$
$$c = (m^e) \bmod n$$

3. Decrypt: $m = K_d(c)$
$$m = \left(c^d\right) \bmod n$$

Example ($e = 11, d = 35, n = 221$)**:**

1. Since $n = 221$, 7 bits ($127 < 221$) segments suffice.

   Plaintext Hi!

   $[100100011010010100001]$

   72 105 33

2. Encrypt $'!'$:
$$(33^{11}) \bmod 221 = 67$$

Cyphertext 67 $\Longrightarrow$ $'C'$

3. Decrypt:

$$(67^{35}) \bmod 221$$
$$= (67^2 \cdot 67^{33}) \bmod 221$$
$$= \{(4489) \bmod 221 \cdot (67^{33}) \bmod 221\} \bmod 221$$
$$= \{69 \cdot (67^{33})\} \bmod 221$$
...
$$= 33 \Longrightarrow '!'$$

Why does RSA work?

- ✓ Key idea

$$m = \left(c^d\right) \bmod n \qquad (1)$$

$$c = (m^e) \bmod n \qquad (2)$$

Substituting (2) to (1)

$$m = \Big(\underbrace{(m^e) \bmod n}_{c}\Big)^d \bmod n \qquad (3)$$

applying

$$\left(c^d\right) \bmod n = \left(c^{d \bmod \phi(n)}\right) \bmod n$$

we have

$$\left(c^{d \bmod \phi(n)}\right) \bmod n = \left((m^e) \bmod n\right)^{d \bmod \phi(n)} \bmod n \Longrightarrow$$

$$\left((m^e) \bmod n\right)^{d \bmod \phi(n)} \bmod n = \left(m^{ed \bmod \phi(n)}\right) \bmod n. \qquad \Longrightarrow$$

$$\left(m^{ed \bmod \phi(n)}\right) \bmod n = (m^1) \bmod n \qquad \Longrightarrow$$

$$\boldsymbol{\left(c^d\right) \bmod n = m}$$

That's why we chose e and d s.t. $(ed - 1) \bmod \phi \equiv 0 \Longrightarrow ed \bmod \phi \equiv 1$

RSA: encryption/decryption (Matlab)

Part 2: Encryption and decryption

```
% Verify the property
disp(['mod(d * e, phi) should be 0: ', num2str(powmod(d*e - 1, 1, phi(p, q)))]);
% Public key is then
disp(['Public key:  (', num2str(e), ',', num2str(n),')']);
disp(['Private key: (', num2str(d), ',', num2str(n),')']);
m = 13; % message
disp(['Message to be encrypted: ', num2str(m)]);
% Encrypt
%c = mod(m^e, n); % won't work, due to the overflow
c = powmod(m, e, n);
disp(['Encrypted message: ', num2str(c)]);
% Decrypt
m_ = powmod(c, d, n);
disp(['Decrypted message: ', num2str(m_)]);
```

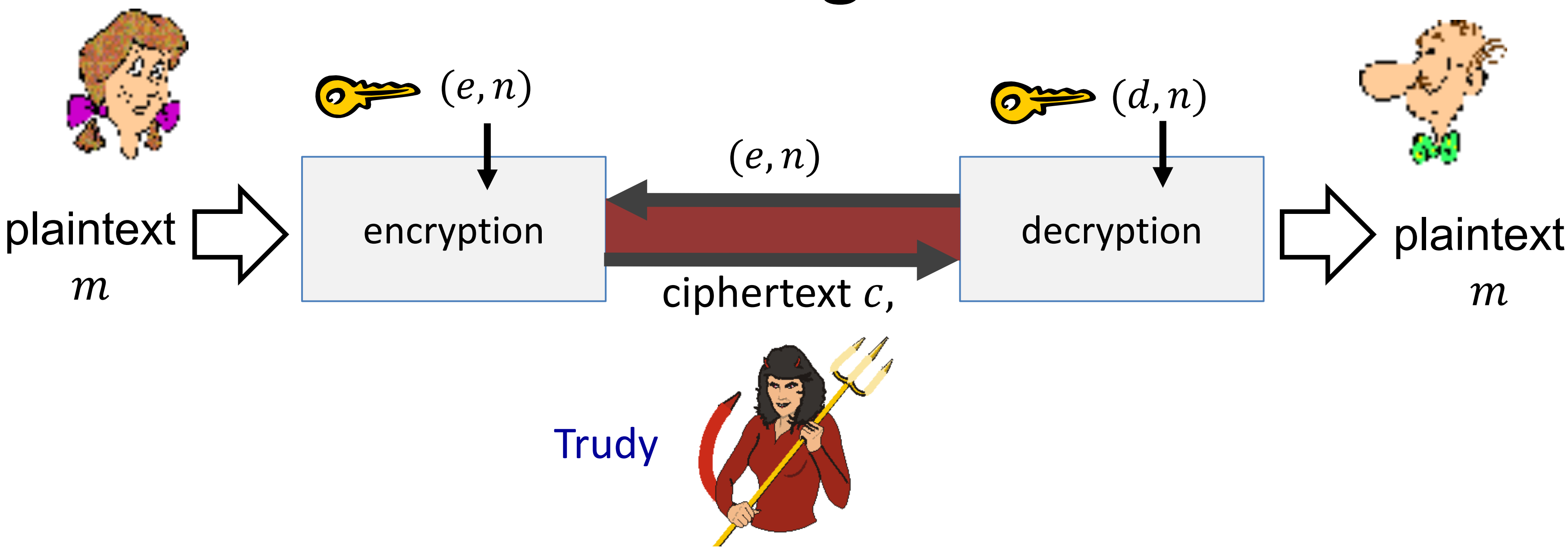
```
function res = powmod(x, e, n)
    res = 1;
    for k = 1:e
        res = mod(res .* x, n);
    end
end
```

Exercises

- **Exercise (paper and pen) 1**: Let the encryption and description keys be
  - (e, n)
  - (d, n)

  where $e = 11, d = 35, n = 221$,
  - Encrypt massages $m_1 = 5, m_2 = 10$ to obtain cyphertexts c_1 and c_2.
  - Decrypt c_1 and c_2 and compare to m_1 and m_2.
- **Exercises (MATLAB) 2**: Let $p = 173$ and $q = 541$
  - Compute (e, n) and (d, n)
  - Encrypt massages $m_1 = 5, m_2 = 10$ to obtain cyphertexts c_1 and c_2.

Cracking RSA



- ✓ Captured (e, n) and c, recover m
- ✓ To decrypt c we need to know d.
- ✓ Recall what numbers are used to compute d?

Cracking RSA

```
% To crack the message
% (1) Factorise n
factors = factor(n);
p_ = factors(1);
q_ = factors(2);
phi_ = (p_ - 1) * (q_ - 1);
% (2) Find a secret key such that mod(d * e, phi_) == 1
d_ = 2;
while(powmod(d_ * e, 1, phi_) ~= 1)
    d_ = d_ + 1;
end
disp(['Recoverd private key: (', num2str(d_), ',', num2str(n),')']);
% (3) Decrypt the message as usual
m_ = powmod(c, d_, n);
disp(['Decrypted message: ', num2str(m_)]);
```

Linear transformations and matrices

- *Matrices* are very useful for describing transformations

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

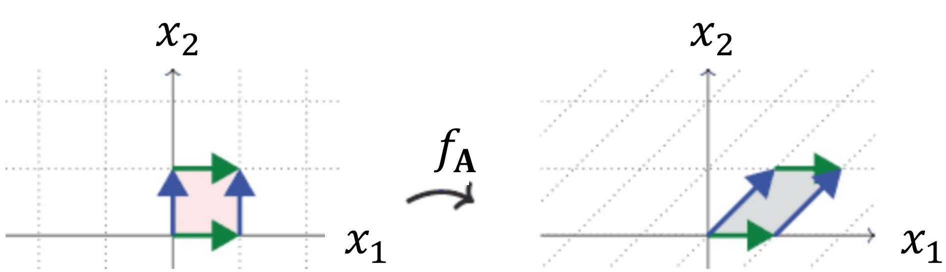
- A plane transformation f can be defined as

$$f_{\mathbf{A}}(\mathbf{v}) = \mathbf{A}\mathbf{s}$$

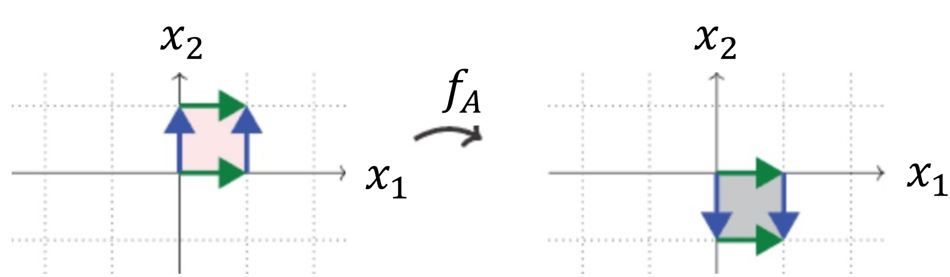
- If $\mathbf{s}$ is the position vector of the point (x_1, x_2) then

$$f_{\mathbf{A}}(\mathbf{s}) = f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \mathbf{A}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} ax_1 + bx_2 \\ cx_1 + dx_2 \end{bmatrix}$$

- **Example**: $\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$



- **Example**: $\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$



Learning with Errors (LWE)

- ✓ Let $\mathbb{Z}_q$ denote the ring of integers modulo q and let $\mathbb{Z}_q^n$ denote the set of *n*-vectors over $\mathbb{Z}_q$.
- ✓ **Example**: $q = 13, n = 4$ and $q = 13, n = 1$

$\mathbf{A} =$

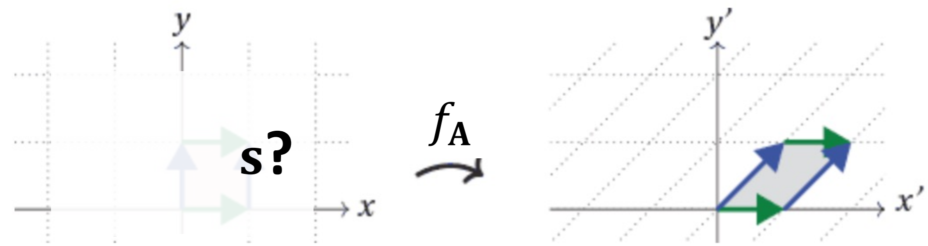
1	1	4	2
5	8	9	5
3	9	0	8
1	3	2	2
7	7	3	4
6	5	12	1
3	3	5	11
$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$

$\mathbf{s} =$

4	y_1
7	y_2
2	y_3
11	y_4
5	y_6
12	y_7
8	y_8

Learning with Errors (LWE)

- ✓ Usually, f_A and $\mathbf{y}$ are known and the problem is to find $\mathbf{s}$ in $f_A\,(\mathbf{s}) = \mathbf{b}$



$\mathbf{A}$

1	1	4	2
5	8	9	5
3	9	0	8
1	3	2	2
7	7	3	4
6	5	12	1
3	3	5	11

$\times$ $\mathbf{s}$ $=$

$\mathbf{b}$

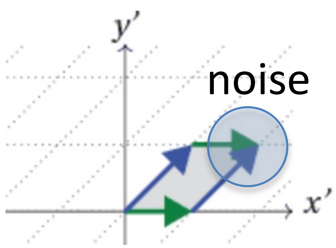
4
7
2
11
5
12
8

known

unknown

- ✓ It is easy to find $\mathbf{s}$.

Learning with Errors (LWE)



1	1	4	2				-1		4
5	8	9	5				0		7
3	9	0	8				0		2
1	3	2	2	×		+	-1	=	11
7	7	3	4				1		5
6	5	12	1				1		12
3	3	5	11				0		8

known

secret

unknown small noise

random: $\mathbf{A} \in \mathbb{Z}_{13}^{7\times4}$ *secret*: $\mathbf{s} \in \mathbb{Z}_{13}^{4\times1}$ *noise*: $\mathbf{e} \in \mathbb{Z}_{13}^{7\times1}$ $\mathbf{b} \in \mathbb{Z}_{13}^{7\times1}$

- ✓ There exists a linear function $f: \mathbb{Z}_q^n \to \mathbb{Z}_q$ and the input to the LWE problem is a sample of pairs $(\mathbf{x}, y)$, where $\mathbf{x} \in \mathbb{Z}_q^n$ and $y \in \mathbb{Z}_q$, so that with high probability $y = f(\mathbf{x})$. The deviation from the equality is according to some known noise model.
- ✓ **Problem**: A hard problem to find $\mathbf{s} \in \mathbb{Z}_{13}^{7\times4}$. [https://en.wikipedia.org/wiki/Learning_with_errors]

Learning with Errors (LWE)

$\mathbf{A} \in \mathbb{Z}_q^{1\times N}$

1. Generate private key

$\mathbf{s} \leftarrow \mathbb{Z}_q^N$

2. Generate public key

$\mathbf{b} = -\mathbf{A}\mathbf{s} + \mathbf{e}$

public key

3. Encrypt message $\mathbf{m}$ to cyphertext $\mathbf{c}$ (each row encrypts one element in $\mathbf{m}$)

$\mathbf{c} = \mathbf{b} + L\mathbf{m}$

ciphertext

4. Decrypt $\mathbf{c}$

$$\mathbf{m} = \frac{1}{L}(\mathbf{A}\mathbf{s} + \mathbf{c})$$

It is easy to see that decryption works

$$\frac{1}{L}(\mathbf{A}\mathbf{s} + \mathbf{c}) = \frac{1}{L}(\mathbf{A}\mathbf{s} + \mathbf{b} + L\mathbf{m}) = \frac{1}{L}(\mathbf{A}\mathbf{s} - \mathbf{A}\mathbf{s} + \mathbf{e} + L\mathbf{m}) = \mathbf{m} + \frac{\mathbf{e}}{L} \approx \mathbf{m}$$

Learning with Errors (LWE)

```
% The value of p can be chosen as a power of 10 such that |m| < p/2 for all messages to be used
env.p = 1e4; % Let the set [p] be where the integer to be encrypted belongs to
env.L = 1e4;
env.r = 1e1;
env.N = 4;  % Number of elements in column vectors in A

sk = Mod( randi(env.p*env.L, [env.N, 1]),  env.p * env.L); % generate secret key
```

```
sk =
   -14106118
   -21444101
   -48662258
    17760247
```

```
m = 30; % message m, also could be a vector
c = encLWE(m,sk,env) % encrypt message m
```

```
c =
   -44887583    29553384    33293629    46706819   -35180159
```

```
m = decLWE(c,sk,env) % decrypt cyphertext c
```

```
m =
     30
```

```
function y = Mod(x,p)
      y = mod(x,p);
      y = y - (y >= p/2)*p; % map [0, p-1] to [-p/2, p/2-1]
end
```

Learning with Errors (LWE)

1. Generate private key

$\mathbf{s} \leftarrow \mathbb{Z}_q^N$

$\mathbf{A} \in \mathbb{Z}_q^{1\times N}$

```
sk = Mod( randi(env.p*env.L, [env.N, 1]),  env.p * env.L); % generate secret key
```

2. Generate public key

$\mathbf{b} = -\mathbf{A}\mathbf{s} + \mathbf{e}$

public key →

$\mathbf{c} = \mathbf{b} + L\mathbf{m}$

← ciphertext

3. Encrypt message $\mathbf{m}$

```
function ciphertext = encLWE(m, sk, env)
    n = length(m);
    q = env.L * env.p; % q = Lp with L being a power of 10
    A = randi(q, [n, env.N]);
    e = Mod(randi(env.r, [n,1]), env.r);
    b = -A*sk + env.L*m + e;
    ciphertext = Mod([b,A], q);
end
```

Learning with Errors (LWE)

4. Decrypt $\mathbf{c}$

$$\mathbf{m} = \frac{1}{L}(\mathbf{A}\mathbf{s} + \mathbf{c})$$

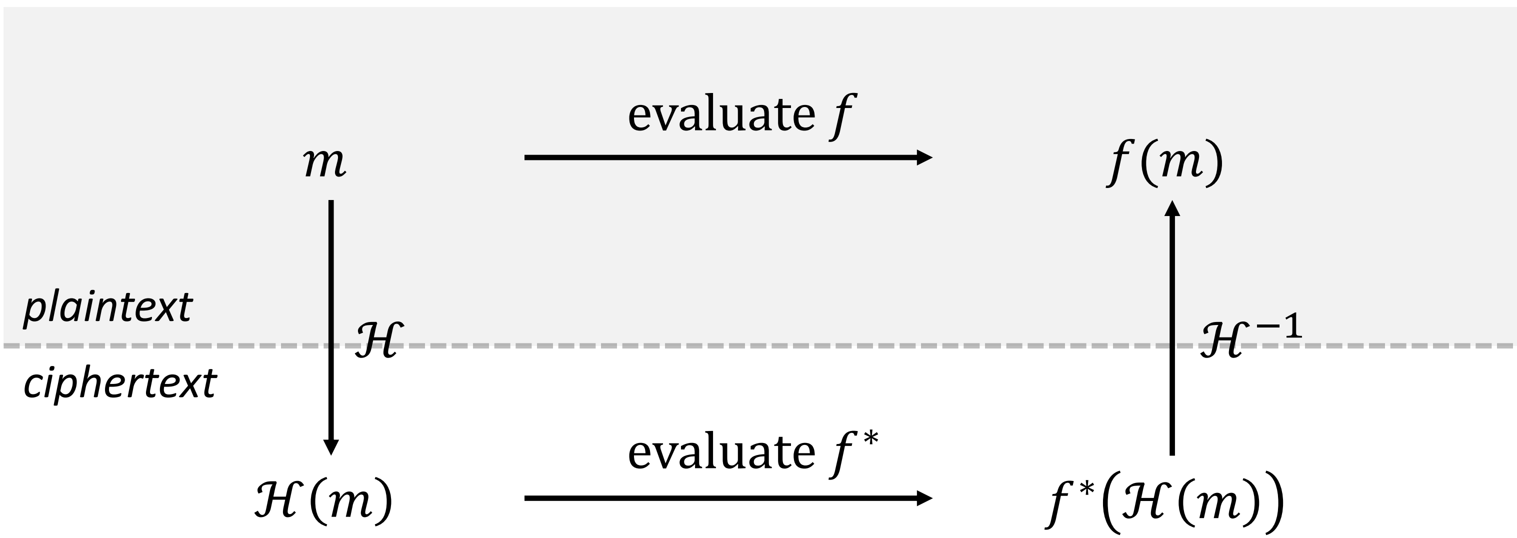
$\mathbf{c} = \mathbf{b} + L\mathbf{m}$

ciphertext

```
function plaintext = decLWE(c,sk,env)
    s = [1; sk];
    plaintext = round( Mod(c*s, env.L*env.p)/env.L );
end
```

- ✓ Recall that `s=[b,A]` so `c*s=[1; sk] * [b,A] = b + sk * A`
- ✓ Recall that `b = −A * sk + env.L * m + e` then
  `c*s = −A * sk + env.L * m + e + sk * A = env.L * m + e`

Homomorphic Encryption as a solution privacy preserving computation



Homomorphic Encryption as a solution privacy preserving computation

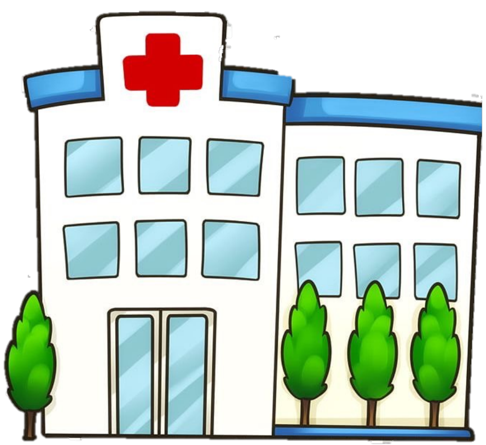
public key

private key

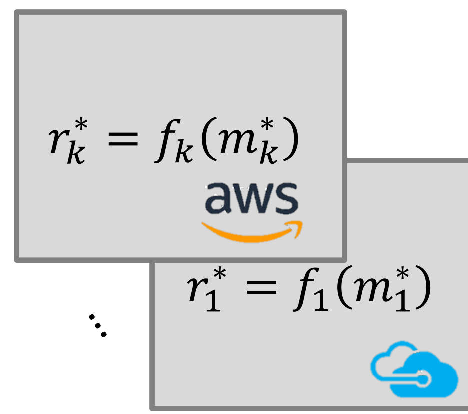
1. Generate Keys (e, d)

2. Encrypt m: $m^* = \mathcal{H}(m, e)$

4. Compute $f : r^* = f(m^*, e)$



3. Send (m^*, e) to the server.



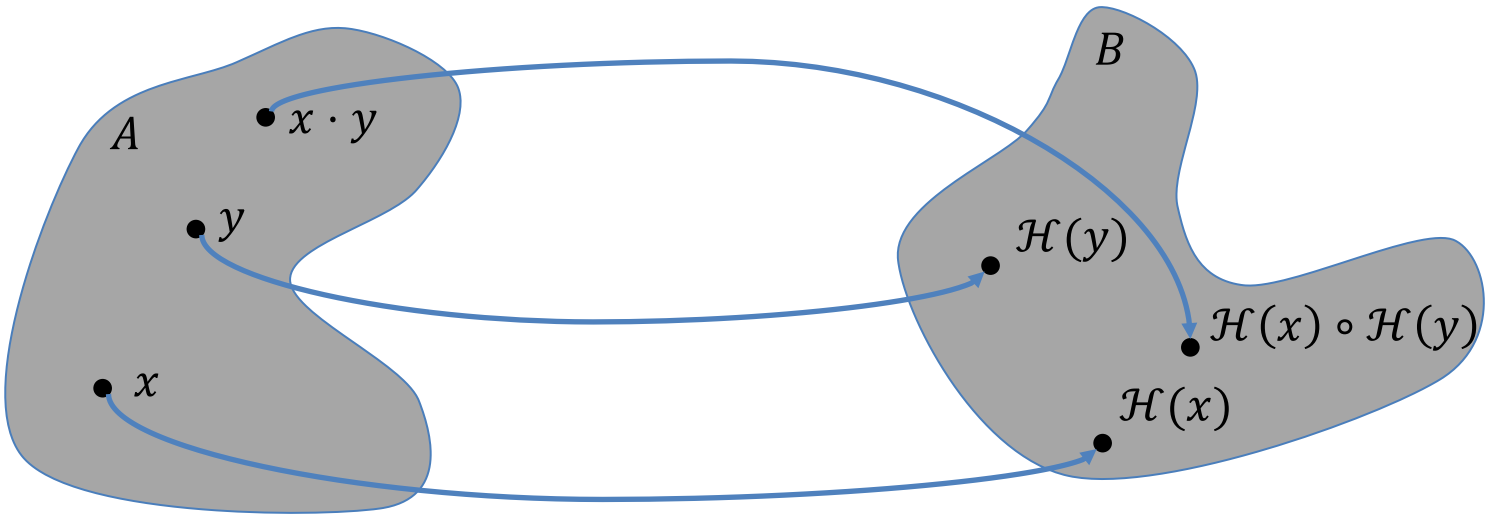
5. Send the result r^* back

6. Decrypt r^*: $r = \mathcal{H}^{-1}(r^*, d)$

What is homomorphism?

- A homomorphism is a *structure-preserving map* between two algebraic structures of the same type.
- A map $\mathcal{H}: A \to B$ between two sets A and B, equipped with the same structure, s.t. if $\cdot$ is an operation of the structure then

$$\mathcal{H}(x \cdot y) = \mathcal{H}(x) \circ \mathcal{H}(y), \forall x, y \in A$$

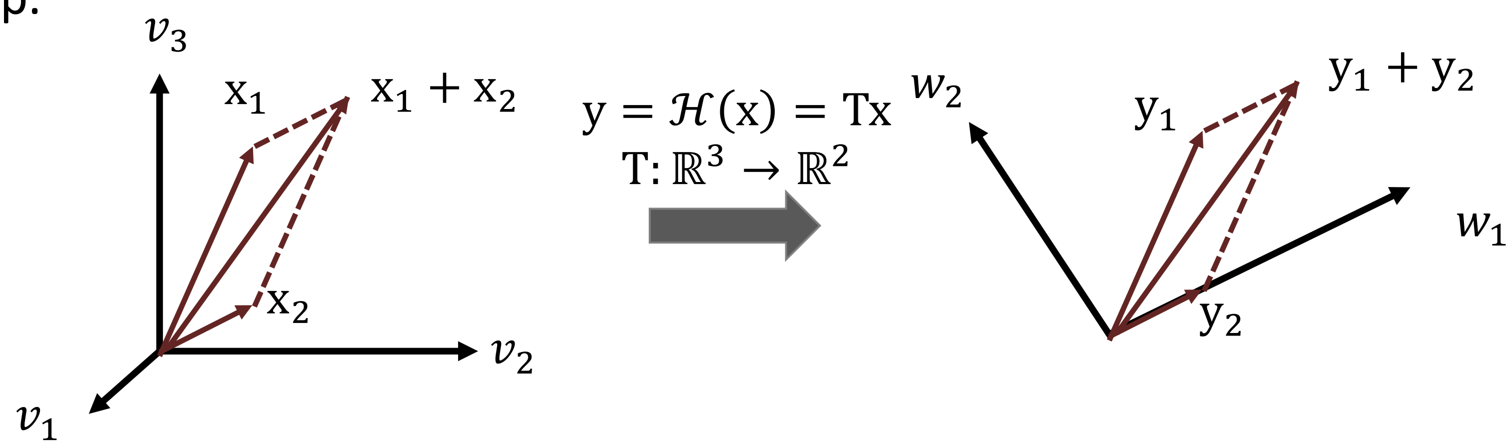


- $\mathcal{H}$ preserves the operation.

What is homomorphism?

- ✓ An algebraic structure may have more than one operation, and a homomorphism is required to preserve each operation.

- ✓ **Example**: A function between vector spaces $\mathcal{H}: \mathcal{V} \rightarrow \mathcal{W}$ that preserves the operations of addition and scalar multiplication is a homomorphism or linear map.



$$\mathcal{H}(\mathrm{x}_1) = T\mathrm{x}_1 = \mathrm{y}_1, \qquad \mathcal{H}(\mathrm{x}_1) = T\mathrm{x}_2 = \mathrm{y}_2$$

$$\mathcal{H}(\mathrm{x}_1 + \mathrm{x}_2) = \mathcal{H}(\mathrm{x}_1) + \mathcal{H}(\mathrm{x}_2) = \mathrm{y}_1 + \mathrm{y}_2, \qquad \mathcal{H}(\alpha \mathrm{x}_1) = \alpha \mathcal{H}(\mathrm{x}_1)$$

$$\mathrm{T}(\mathrm{x}_1 + \mathrm{x}_2) = \mathrm{Tx}_1 + \mathrm{Tx}_2 = \mathrm{y}_1 + \mathrm{y}_2\,, \qquad \mathrm{T}(\alpha \mathrm{x}_1) = \alpha \mathrm{Tx}_1$$

What is homomorphism?

- The notation for the operations does not need to be the same in the source and the target of a homomorphism.
- **Group homomorphism**: Given two groups $(G,*)$ and $(H,\cdot)$, a function $\mathcal{H}: G \to H$ is group homomorphism if

$$\mathcal{H}(x * y) \to \mathcal{H}(x) \cdot \mathcal{H}(y), \qquad \forall x, y \in G$$

- **Example**:

$$\mathcal{H}: x \to e^x, \forall x \in \mathbb{R}$$
$$(\mathbb{R},+) \overset{\mathcal{H}}{\to} (\mathbb{R}^+,\cdot)$$
$$\mathcal{H}(x+y) = \mathcal{H}(x)\mathcal{H}(y) \Longrightarrow$$
$$e^{x+y} = e^x e^y,$$

$$\mathcal{G}: x \to \ln x, \forall x \in \mathbb{R}^+$$
$$(\mathbb{R}^+,\cdot) \overset{\mathcal{G}}{\to} (\mathbb{R},+)$$
$$\mathcal{G}(xy) = \mathcal{G}(x) + \mathcal{G}(y) \Longrightarrow$$
$$\ln(xy) = \ln(x) + \ln(y)$$

$\mathcal{H}$ is also an *isomorphism* as its inverse function $\mathcal{H}^{-1} = \mathcal{G}(x)$ forms a group homomorphism

- **Example**: Compute $f(x) = 2x + 1$, on "encrypted x" $\mathcal{H}(x)$

$$\mathcal{H}\{2x+1\} = e^{2x+1} = e^x e^x e = \mathcal{H}(x)\mathcal{H}(x)\mathcal{H}(1) = \left(\mathcal{H}(x)\right)^2 \cdot \mathcal{H}(1)$$

$$\mathcal{G}\left\{\left(\mathcal{H}(x)\right)^2 \cdot \mathcal{H}(1)\right\} = \mathcal{G}\left\{\left(\mathcal{H}(x)\right)^2\right\} + \mathcal{G}\{\mathcal{H}(1)\} = \mathcal{G}\{\mathcal{H}(x)\} + \mathcal{G}\{\mathcal{H}(x)\} + 1 = 2x + 1$$

Example of multiplicative homomorphism using RSA

✓ RSA scheme is multiplicatively homomorphic

Compute $m_1 m_2$:

1. Generate keys (e, n) and (d, n)
2. Encrypt:
   $$c_1 = \mathcal{H}_e(m_1) = (m_1^e) \bmod n$$
   $$c_2 = \mathcal{H}_e(m_2) = (m_2^e) \bmod n$$
3. Compute:
   $$c_1 c_2 = \mathcal{H}_e(m_1) \cdot \mathcal{H}_e(m_2)$$
   $$= \left((m_1^e) \bmod n\right) \cdot \left((m_1^e) \bmod n\right)$$
   $$= (m_1 m_2)^e \bmod n = \mathcal{H}_e(m_1 m_2) = c_{12}$$
4. Decrypt: c_{12}
   $$\mathcal{H}_d(c_{12}) = \left(c_{12}^d\right) \bmod n = m_1 m_2$$

Example:

1. Let $e = 11, d = 35, n = 221$, and $m_1 = 5, m_2 = 10$
2. Encrypt :
   $$c_1 = (5^{11}) \bmod 221 = 164$$
   $$c_2 = (10^{11}) \bmod 221 = 173$$
3. Compute
   $$c_{12} = 164 \cdot 173 = 28372$$
4. Decrypt:
   $$m_1 m_2 = (28372^{35}) \bmod 221$$

$m_1 m_2 = ($
7098200968290592840991958652267788
1571486384800862824462878933961928
9085294896750081950091846259916085
3592398936486064467237262654024462
2619997701833280716$8) \bmod 221 = \mathbf{50}$

Example of multiplicative homomorphism (MATLAB)

```
% Define the keys
e = 11; d = 35; n = 221;
% Display public and private keys
disp(['Public key: (', num2str(e), ',', num2str(n),')']);
disp(['Private key: (', num2str(d), ',', num2str(n),')']);
% Define the numbers
m1 = 5;
m2 = 10;
disp(['Message to be encrypted: ', num2str(m1)]);
disp(['Message to be encrypted: ', num2str(m2)]);
% Encrypt the numbers
c1 = powmod(m1, e, n);
c2 = powmod(m2, e, n);
disp(['Encrypted number1: ', num2str(c1)]);
disp(['Encrypted number2: ', num2str(c2)]);
% Compute (multiply) over encrypted numbers
c12 = c1 * c2;
disp(['Encrypted results: ', num2str(c12)]);
% Decrypt the result
m_ = powmod(c12, d, n);
disp(['Decrypted message: ', num2str(m_)]);
```

```
Public key:  (11,221)
Private key: (35,221)
Message to be encrypted: 5
Message to be encrypted: 10
Encrypted number1: 164
Encrypted number2: 173
Encrypted results: 28372
Decrypted message: 50
```