Week 5: Introduction to Cryptography
One-Time Pad, Perfect Secrecy,
and Public-Key Cryptography (Diffie-Hellman)
(Textbook Chapter 6)

Prof. Michael Gastpar

Slides by Prof. M. Gastpar and Prof. em. B. Rimoldi

EPFL

Spring Semester 2025

Outline

J,DBNF.,J,TBX.,J,DPORVFSFEA

Cryptography serves two purposes:

- Privacy: Preventing that sensitive information lands in the wrong hands.
- Authenticity: Preventing that information is falsified.

Before the Internet:

- Cryptography was essentially a tool for diplomats and generals.
- Common people would sign a letter (for authenticity), put it in an envelope (for privacy) and trust the postal service for the delivery to the intended recipient (reliability).

The Internet has changed that:

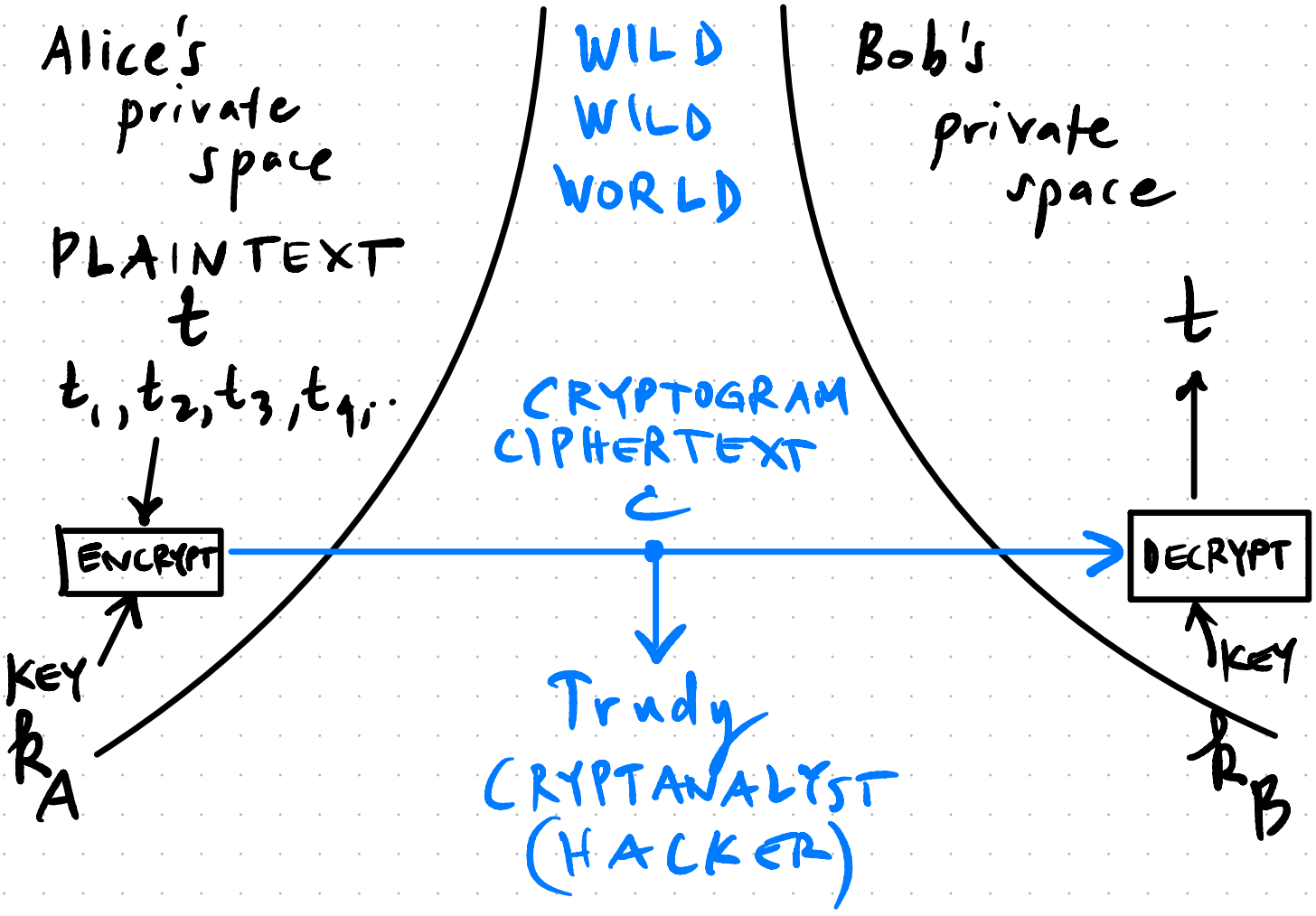
- Now we send sensitive information over public channels on a daily basis. We need to control who can decipher such information (privacy). People and businesses can be destroyed if private information leaks out.

- We have the ability to post information that can be read by anybody — hence that can have a huge impact. We need to be able to verify who is posting (authenticity). People and businesses can be destroyed if information is falsified.

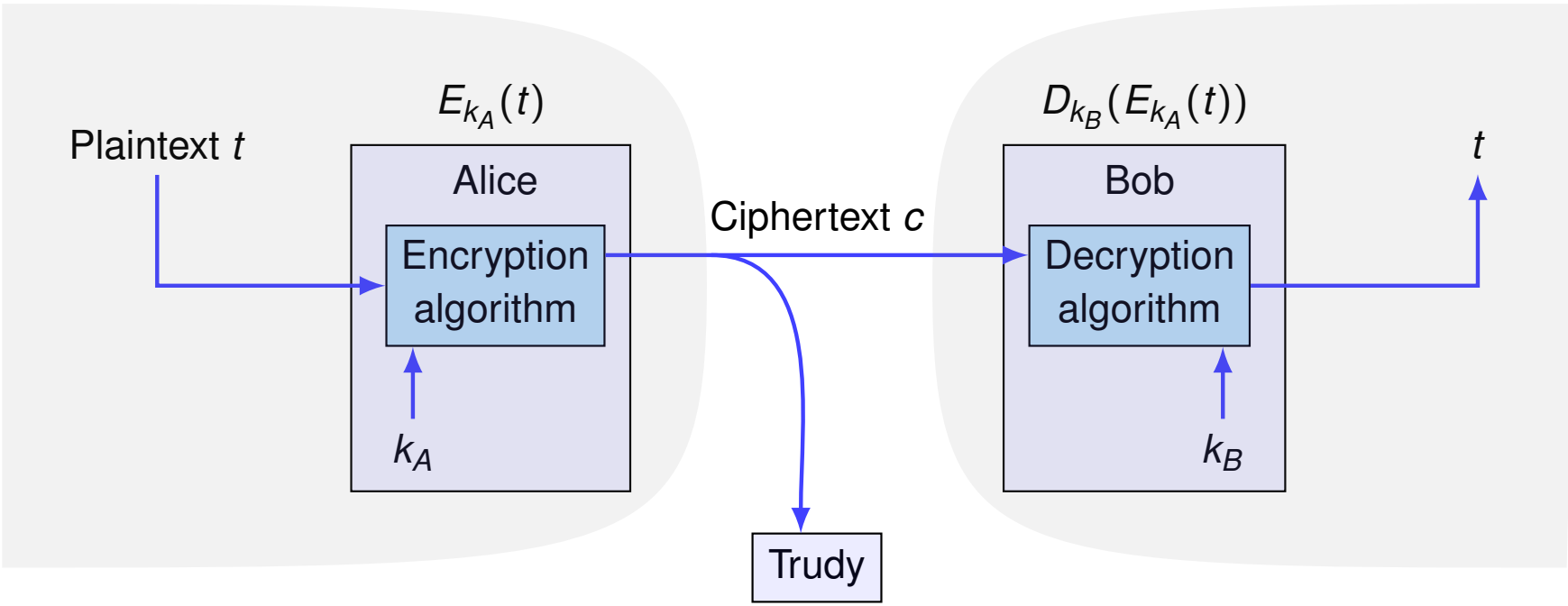
Cryptography gives us the tools to:

- authenticate the sender and the receiver
- verify the integrity of the message
- keep the message confidential

All these problems are related. Our initial focus is on how to keep a message confidential.


Alice's private space
WILD WILD WORLD
Bob's private space
PLAINTEXT
t
$t_1, t_2, t_3, t_4, \ldots$
CRYPTOGRAM
CIPHERTEXT
c
t
ENCRYPT
DECRYPT
KEY
k_A
KEY
k_B
Trudy
CRYPTANALYST
(HACKER)

BASIC SETUP FOR CONFIDENTIALITY



Alice wants to sent the plaintext t to Bob:

- She encrypts t using her key k_A. The result is the ciphertext $c = E_{k_A}(t)$.
- She sends c to Bob over a public channel.
- Bob decrypts c using his key k_B. The result is $D_{k_B}(E_{k_A}(t)) = t$.
- For Trudy, it is nearly impossible to recover t from c without knowing k_B.

- plaintext, ciphertext (also called cryptogram), key, encrypter, decrypter: already defined.
- cryptography: the art of composing cryptograms.
- cryptanalysis: the art of breaking cryptograms.
- a cryptanalyst has broken the system when he can quickly determine the plaintext from the cryptogram, no matter what key is used.
- attacker: same as cryptanalyst.

CAESAR'S CYPHER

SPACE

A	B	C	D	E	…	W	X	Y	Z	␣	,	.
↓	↓	↓	↓	↓		↓	↓	↓	↓	↓	↓	↓
D	E	F	G	H	…	Z	␣	,	.	A	B	C

CAESAR'S CYPHER

MODULO perspective

A	B	C	D	E	...	W	X	Y	Z	␣ (SPACE)	,	.
↓	↓	↓	↓	↓		↓	↓	↓	↓	↓	↓	↓
0	1	2	3	4	--	22	23	24	25	26	27	28

CAESAR'S CYPHER

MODULO perspective

SPACE

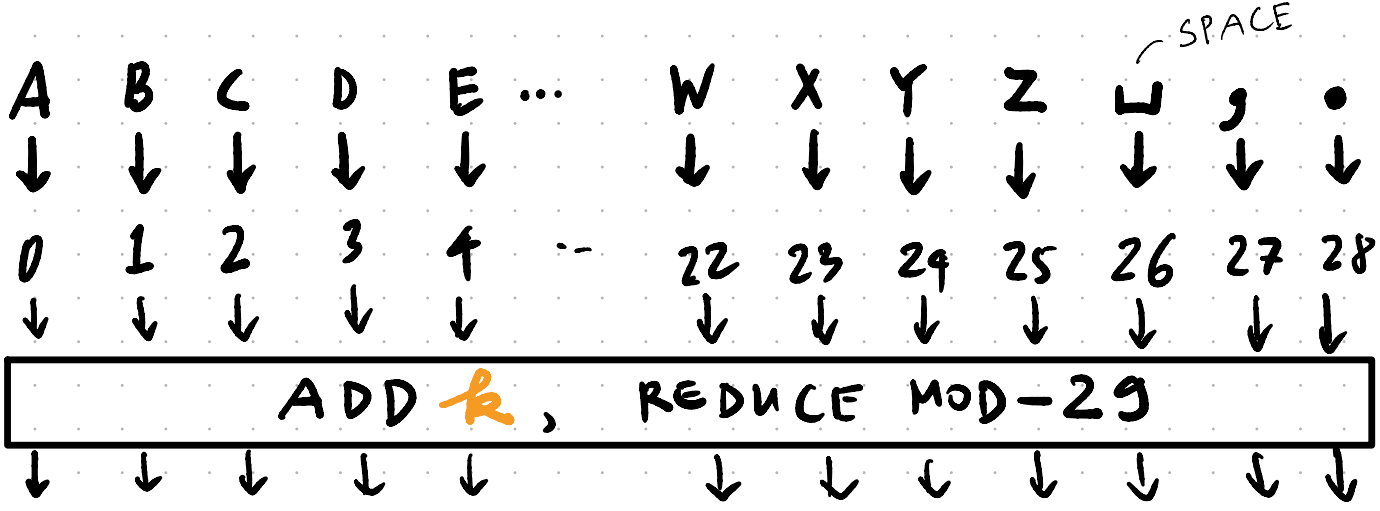
A	B	C	D	E	...	W	X	Y	Z	␣	,	.
↓	↓	↓	↓	↓		↓	↓	↓	↓	↓	↓	↓
0	1	2	3	4	...	22	23	24	25	26	27	28
↓	↓	↓	↓	↓		↓	↓	↓	↓	↓	↓	↓

ADD 3, REDUCE MOD-29

↓	↓	↓	↓	↓		↓	↓	↓	↓	↓	↓	↓
3	4	5	6	7	...	25	26	27	28	0	1	2

CAESAR'S CYPHER

MODULO perspective



TRY $k=1$

J,DBNF.,J,TBX.,J,DPORVFSFEA

↓

INCAME

Caesar's Cipher (Julius Caesar (1st century BC))

Suppose that we are using the English alphabet augmented by a few special characters, say "space", "comma", and "period".

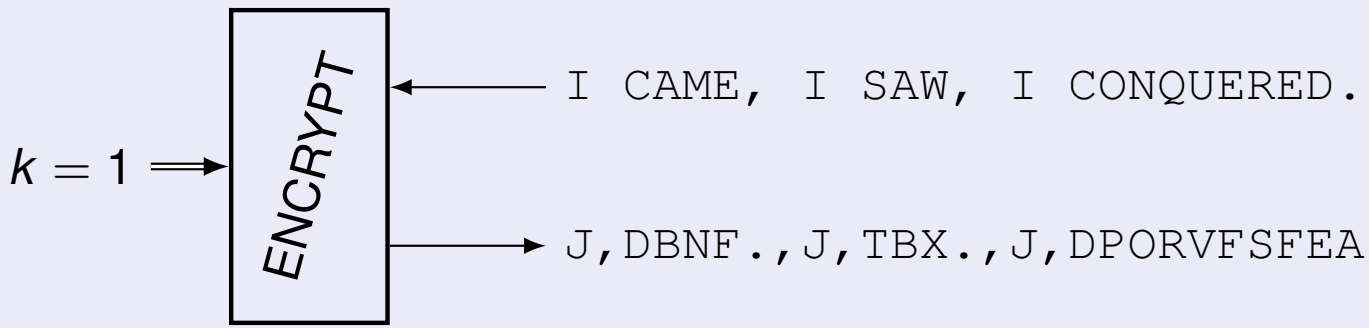
An alphabet of 29 characters, represented by the integers $0, 1, \ldots, 28$.

- the key k is an integer between 0 and 28, known to Alice and Bob and to nobody else.
- the encryption algorithm substitutes the i-th letter of the alphabet with the $(i + k)$-th letter $\pmod{29}$.
- the decryption algorithm substitutes the j-th letter with the $(j - k)$-th $\pmod{29}$.

Example (Caesar's cipher)

The alphabet is

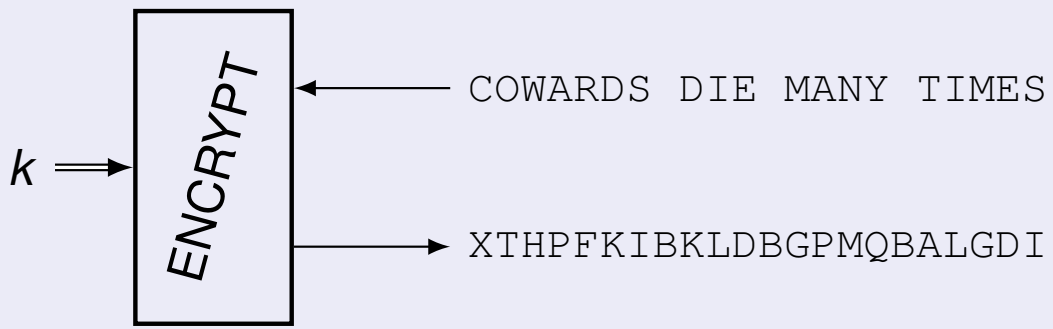
$$\{A, B, C, D, E, \ldots, W, X, Y, Z, \text{space}, \text{comma}, \text{period}\}$$



Monoalphabetic Cipher

Caesar's cipher is a special case of a monoalphabetic cipher. A more general monoalphabetic cipher uses an arbitrary permutation of the alphabet.

Example (Monoalphabetic cipher)



k
A → P
B → V
C → X
D → K
E → D
F → C
G → O
H → J
I → L
J → W
K → Z
L → E
M → G
N → M
O → T
P → Y
Q → S
R → F
S → I
T → A
U → N
V → U
W → H
X → R
Y → Q
Z → space
space → B

Polyalphabetic Cipher

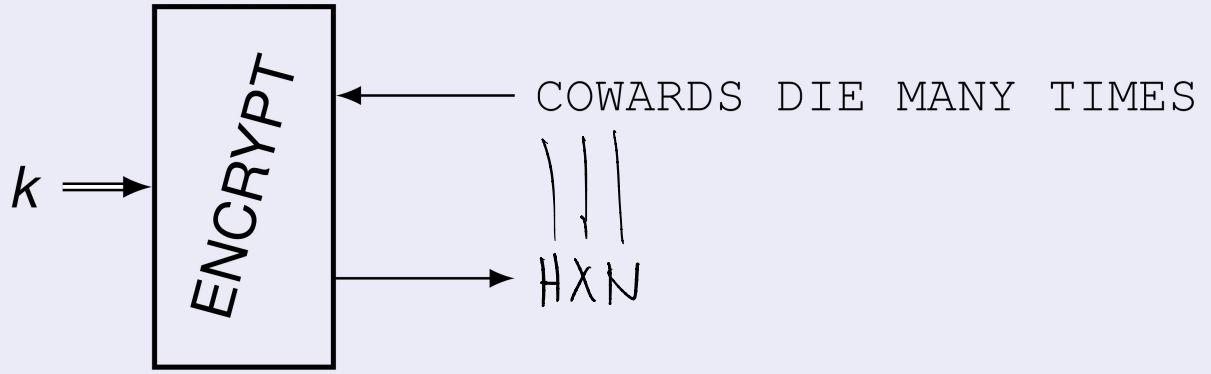
A monoalphabetic cipher uses a fixed substitution table over the entire message. A polyalphabetic cipher uses multiple substitution tables.

A key specifies which table is used for which position of the message.

Example (Polyalphabetic cipher: Vigenère's cipher)

- It uses multiple Caesar ciphers.
- So if the key is 5,9,20, it means
  - the offset for the first letter of the message is 5
  - that for the second letter is 9
  - for the third letter it is 20
  - for the fourth letter it is 5 (we start over with the first offset of the key)
  - etc.

VIGENÈRE : EXAMPLE



$k = 5, 9, 20$

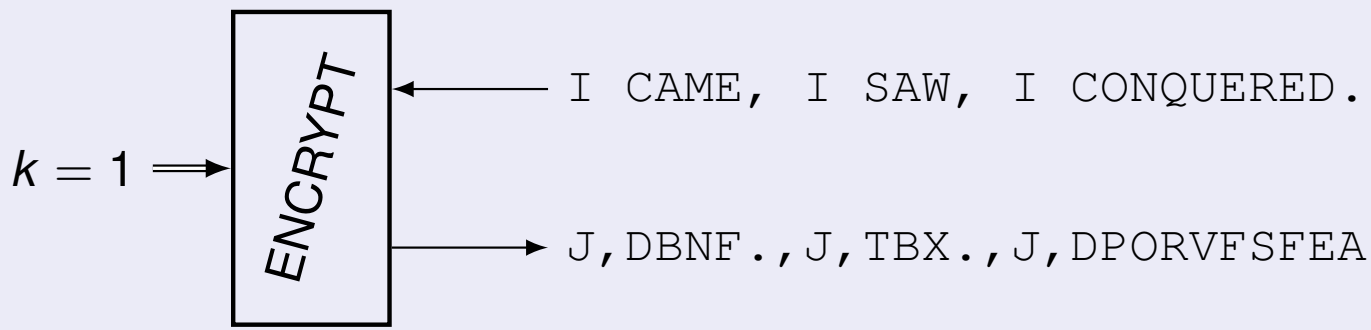
$\{A, B, C, D, E, \ldots, W, X, Y, Z, \text{space}, \text{comma}, \text{period}\}$

Key Assumption In Modern Cryptography

The security is based on the secret key (not on the secrecy of the algorithm).

Example (counterexample)

Caesar was evidently relying on the secrecy of the algorithm.



We distinguish between the following attacks:

- ciphertext-only: one or more cryptograms available to the cryptanalyst, known to have been encrypted with the same key.
- known plaintext: the cryptanalyst has one or more plaintexts and the resulting cryptograms, known to have been encrypted with the same key.
- chosen plaintext: for any plaintext that he requires, the cryptanalyst can obtain the cryptogram under the same key.

What kind of security do we expect?

- Ideally, a cryptographic system should be secure against a chosen plaintext attack.
- At the very least, it should be secure against a ciphertext-only attack.

Example (Caesar's cipher)

- chosen plaintext attack: encrypt one letter and you get the key
- known plaintext attack: compare one letter and get the key
- ciphertext-only attack: try all the 29 possible keys

Caesar's cipher is not at all secure against a contemporary attacker.

Example (Generic Monoalphabetic cipher)

- chosen plaintext attack: encrypt each letter of the alphabet
- known plaintext attack: compare input/output over a text that uses all letters
- ciphertext-only attack:
  - brute-force approach: try all $29! = 8.84 \times 10^{30}$ permutations
  - letter-frequency approach: use the fact that for a given language we know the frequency of each letter

A brute-force approach is challenging.

With a modern computer, the key can easily be found using the letter-frequency attack.

How to make the letter-frequency attack unfruitful?

Example (Vigenère's cipher, with an n-length key)

- chosen plaintext attack: encode the same letter until you have the n-length key
- known plaintext attack: compare input/output until you have the n-length key
- ciphertext-only attack:
  - brute-force approach: try all 29^n keys if you know n. (Many more otherwise.)
    - for $n = 21$, the number of keys is 5.13^{30}
    - for $n = 100$, the number of keys is 1.73^{146}
  - if you know n, you can partition input/output into n parts, each of which is a Caesar cipher with its own key.
  - letter-frequency approach: effective if the plaintext-length to key-length ratio is sufficiently large.

HOMEWORK 5, PROBLEM 1

EBGRYCXGBGHITURSYNEAVCGBGRYV

HOMEWORK 5, PROBLEM 1

EBGRYCXGBGHITURSYNEAVCGBGRYV

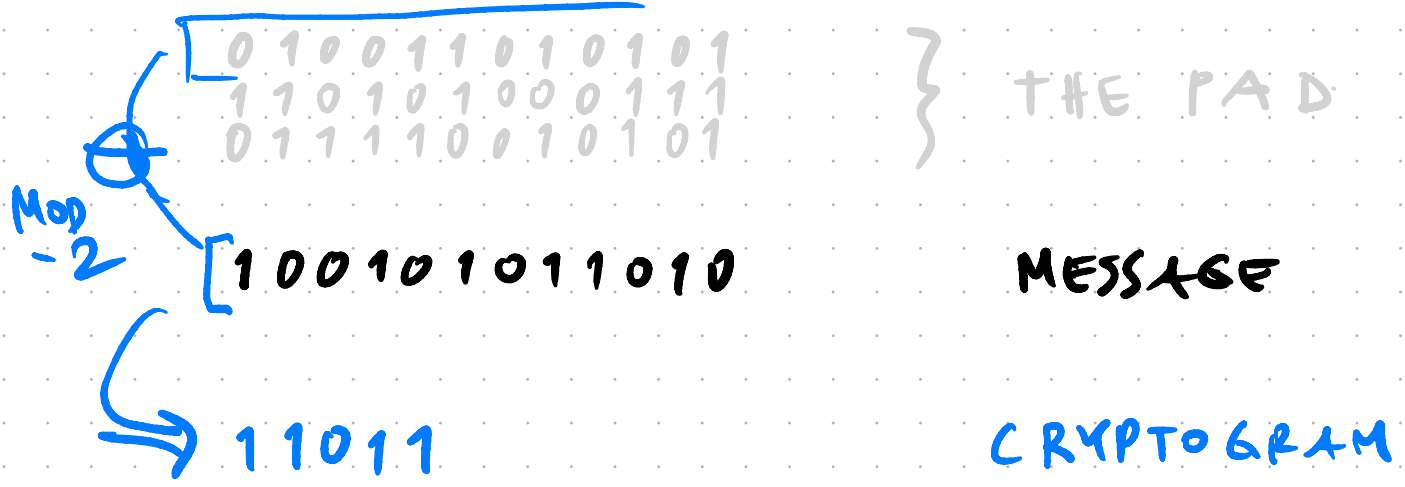
T**********************U**I**I***

THE ONE-TIME PAD

.1001

} THE PAD

THE ONE-TIME PAD



Preliminary assumptions:

- The plaintext t, the key k and the cryptogram c are n-length binary sequences over the alphabet $\mathcal{A} = \{0, 1\}$.
- The key k is produced by selecting each bit independently and with uniform distribution.
- Alice and Bob use a private channel to exchange the key ahead of time.

Encryption: $c = t \oplus k$ (component-wise binary sum)

Decryption: $c \oplus k = (t \oplus k) \oplus k = t \oplus (k \oplus k) = t$

Example (one-time pad)

Encryption:

t	=	1	0	1	1	0	1
k	=	0	1	0	0	0	1
c	=	1	1	1	1	0	0

Decryption:

c	=	1	1	1	1	0	0
k	=	0	1	0	0	0	1
t	=	1	0	1	1	0	1

We get back the plaintext because $b + b = 0 \pmod 2$ for $b \in \{0, 1\}$.

Generalizing to a non-binary alphabet is straightforward.

Example (one-time pad)

Encryption:

t	=	1	0	1	1	0	1
k	=	0	1	0	0	0	1
c	=	1	1	1	1	0	0

Decryption:

c	=	1	1	1	1	0	0
k	=	0	1	0	0	0	1
t	=	1	0	1	1	0	1

We get back the plaintext because $b + b = 0 \pmod 2$ for $b \in \{0, 1\}$.

Generalizing to a non-binary alphabet is straightforward.

GENERAL ONE-TIME PAD

$\mathcal{A} = \{0, 1, 2, 3, 4\}$

$t = 3\ 4\ 0\ 2\ 1\ 2$

$k = 1\ 2\ 0\ 3\ 2\ 4$

UNIFORMLY INDEPENDENT

$C = 4\ 1\ 0\ 0$

DECRYPT?
→ SUBTRACT k.

GUESSING

X: RANDOM VARIABLE.

$p(X=1) = 1/2$

$p(X=2) = 1/4$

$p(X=3) = 1/4$.

GUESS X ?

PICK $X=1$.

GUESSING

X: RANDOM VARIABLE.

$p(X=1) = 1/3$

$p(X=2) = 1/3$

$p(X=3) = 1/3$

GUESS X ?

DOESN'T MATTER

WHAT YOU PICK.

GUESSING

X : RANDOM VARIABLE.

Y : SECOND R.V.

$P(X\|Y)$	$Y=0$	$Y=1$
$X=0$	$1/2$	$1/4$
$X=1$	$1/4$	$1/4$
$X=2$	$1/4$	$1/2$

ONLY ONE GUESS, KNOWING Y.

$Y=0$: GUESS $X=0$

$Y=1$: GUESS $X=2$

GUESSING

X : RANDOM VARIABLE.

Y : SECOND R.V.

$P(X\|Y)$	$Y=0$	$Y=1$
$X=0$	$1/2$	$1/2$
$X=1$	$1/4$	$1/4$
$X=2$	$1/4$	$1/4$

ONLY ONE GUESS, KNOWING Y.

↳ GUESS $X=0$

(IGNORE Y)

Y IS USELESS HERE.

OBSERVATION:

IF X AND Y ARE INDEPENDENT,

THEN Y DOES NOT HELP IN GUESSING X.

DEFINITION (PERFECT SECRECY)

A cryptosystem has **perfect secrecy** if the plaintext T and the cryptogram C are statistically independent.

Perfect secrecy is the ultimate kind of security against a ciphertext-only attack: The attacker cannot do better than guessing the plaintext T.

- The n-length key k is selected at random (uniform distribution over $\{0,1\}^n$).

← n bits of message

- The key k and the message t are selected independently.

- The ciphertext is $c = t \oplus k$.

$$p_{C|T}(c|t) = p_{K|T}(c \ominus t|t) = p_K(c \ominus t) = \frac{1}{2^n}.$$

(n is known by assumption.)

Hence C and T are independent: knowledge of C is useless in guessing T.

$n = 5$:

$$P_{C|T}(10010 \mid 01110)$$

$$= P_{KEY}(11100) = \frac{1}{32}.$$

A Weakness of the one-time pad

Example (one-time pad)

An cryptanalyst that has the plaintext t and the corresponding cryptogram c, immediately gets the key:

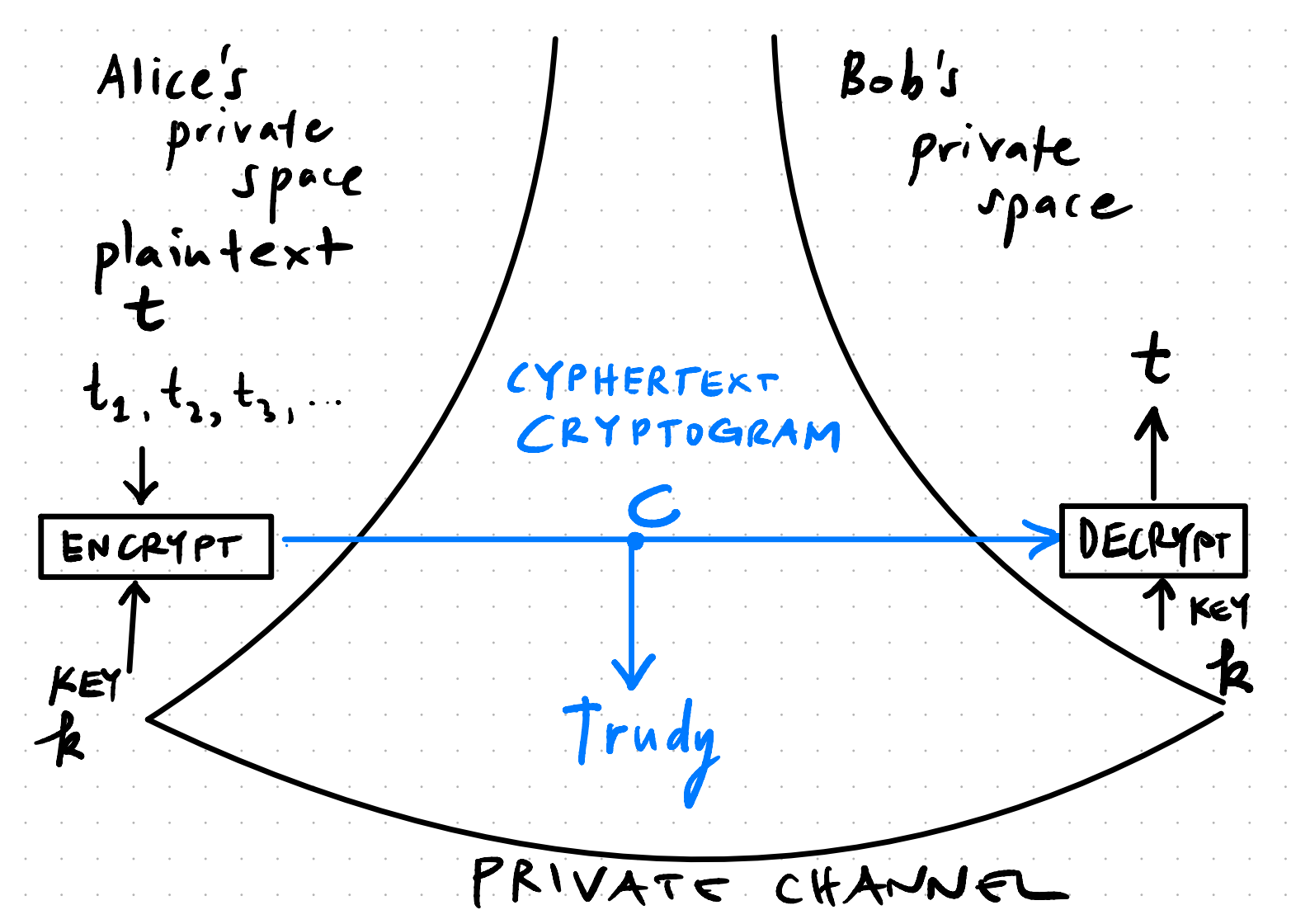
$$k = c \ominus t$$

Hence the pad (the key) should be used only once.

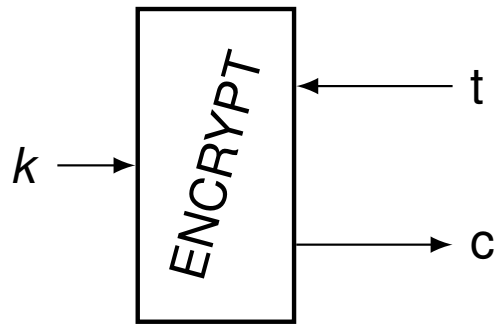
ONE-TIME PAD: ADVANTAGES AND DRAWBACKS

+ very simple algorithm
+ as secure as it gets against a ciphertext-only attack and key used once
+ of instructional value to prove that perfect secrecy is possible
- the key is as long as the plaintext (this is fundamental, see later)
- the key needs to be exchanged ahead of time over a private channel
- a ciphertext-only attack can break the system if the key is used twice (see homework)
- a known plaintext attack reveals the key

The "one-time pad" has been used extensively in diplomatic and espionage circles.


Alice's private space
plaintext
t
$t_1, t_2, t_3, \ldots$
ENCRYPT
KEY
k
CYPHERTEXT
CRYPTOGRAM
c
Trudy
Bob's private space
t
DECRYPT
KEY
k
PRIVATE CHANNEL

The following Theorem makes no assumption on the encryption algorithm.



THEOREM (PERFECT SECRECY)

Perfect secrecy implies

$$H(T) \leq H(K).$$

Thm: IF PERFECT SECRECY, $\iff T \perp C$

THEN $H(T) \leq H(K)$.

PROOF: T, K, C

$$H(T,K,C) = H(C) + \underbrace{H(T|C)}_{=H(T)} + H(K|T,C)$$

$$= H(C) + H(K|C) + \underbrace{H(T|K,C)}_{=0}$$

$$\Rightarrow H(T) + H(K|T,C) = H(K|C)$$

$$\Rightarrow \quad H(T) + \underbrace{H(K|T,C)}_{\geq 0} \leq H(K)$$

$$\Rightarrow \quad H(T) \leq H(K) \qquad \square$$

Thm: IF PERFECT SECRECY,
THEN $H(T) \leq H(K)$.

PROOF:

$$H(T,K,C) = H(C) + \underbrace{H(T|C)}_{=H(T)} + H(K|T,C)$$

$$= H(C) + H(K|C) + \underbrace{H(T|K,C)}_{=0}$$

$$\Rightarrow H(T) \leq H(K|C)$$

Thm: IF PERFECT SECRECY, THEN $H(T) \leq H(K)$.

PROOF:

$$H(C) + H(T)$$

$$= H(C) + H(T|C)$$ — $T \perp C$

$$= H(C,T)$$ — CHAIN RULE

$$\leq H(C,T,K)$$ — CHAIN RULE & NON-NEG OF ENTROPY

$$= H(C) + H(K|C) + H(T|K,C)$$ — CHAIN RULE

Proof:

Perfect secrecy $\big(H(T) = H(T|C)\big)$ and decodability $\big(H(T|K,C) = 0\big)$ imply

$$\begin{aligned} H(T) &= H(T|C) \\ &\leq H(T,K|C) \\ &= H(K|C) + H(T|K,C) \\ &= H(K|C) \\ &\leq H(K). \end{aligned}$$

□

NB: Entropy plays a key role also in cryptography.

EXERCISE

Determine the minimum average length of the binary key for a cryptosystem that has the following characteristics:

- the message is an uncompressible binary string of length n
- the system achieves perfect secrecy

Solution

- $H(T)$ must be (essentially) n bits (otherwise further compression is possible).
- perfect secrecy requires $H(T) \leq H(K)$.
- hence $H(K)$ is at least n.
- the average blocklength of the binary key is at least n bits.

A symmetric-key cryptosystem is one for which both ends use the same key ($k_A = k_B = k$). All examples considered so far rely on a symmetric key.

There exists fast (and secure) symmetric-key cryptosystems, but:

- Anybody that has the key can encrypt and/or decrypt.
- The key cannot be sent over an insecure channel.
- In an n-user network, each user needs $n - 1$ keys to communicate privately with every other user. Key distribution is a problem as it has to be done over a secure channel. And keys have to be changed frequently!
- We have a real problem: see e.g. the first 6 min. and 20 sec. of `http://www.youtube.com/watch?v=YEBfamv-_do&sns=em`

Public Key-Distribution (Diffie and Hellman)

Is there a way to distribute keys over a public channel?

In 1976, Diffie and Hellman came up with a solution.

EXAMPLE: $p = 7$: $\mathcal{A} = \{0, 1, 2, 3, 4, 5, 6\}$

$g = 3$

i	$g^i \bmod 7$
0	
1	3
2	2
3	6
4	4
5	5
6	1

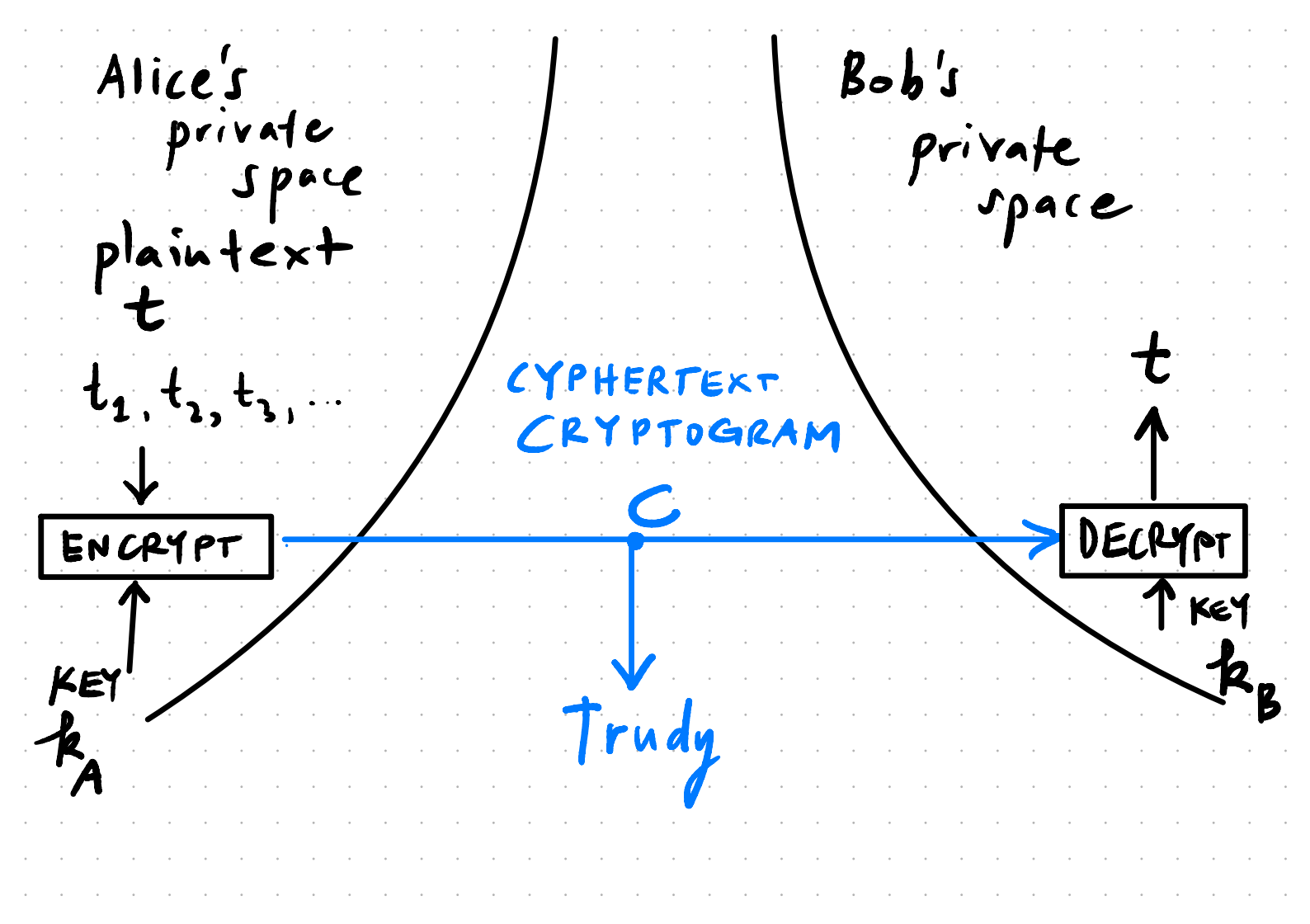
EXAMPLE: $p = 7$: $\mathcal{A} = \{0, 1, 2, 3, 4, 5, 6\}$

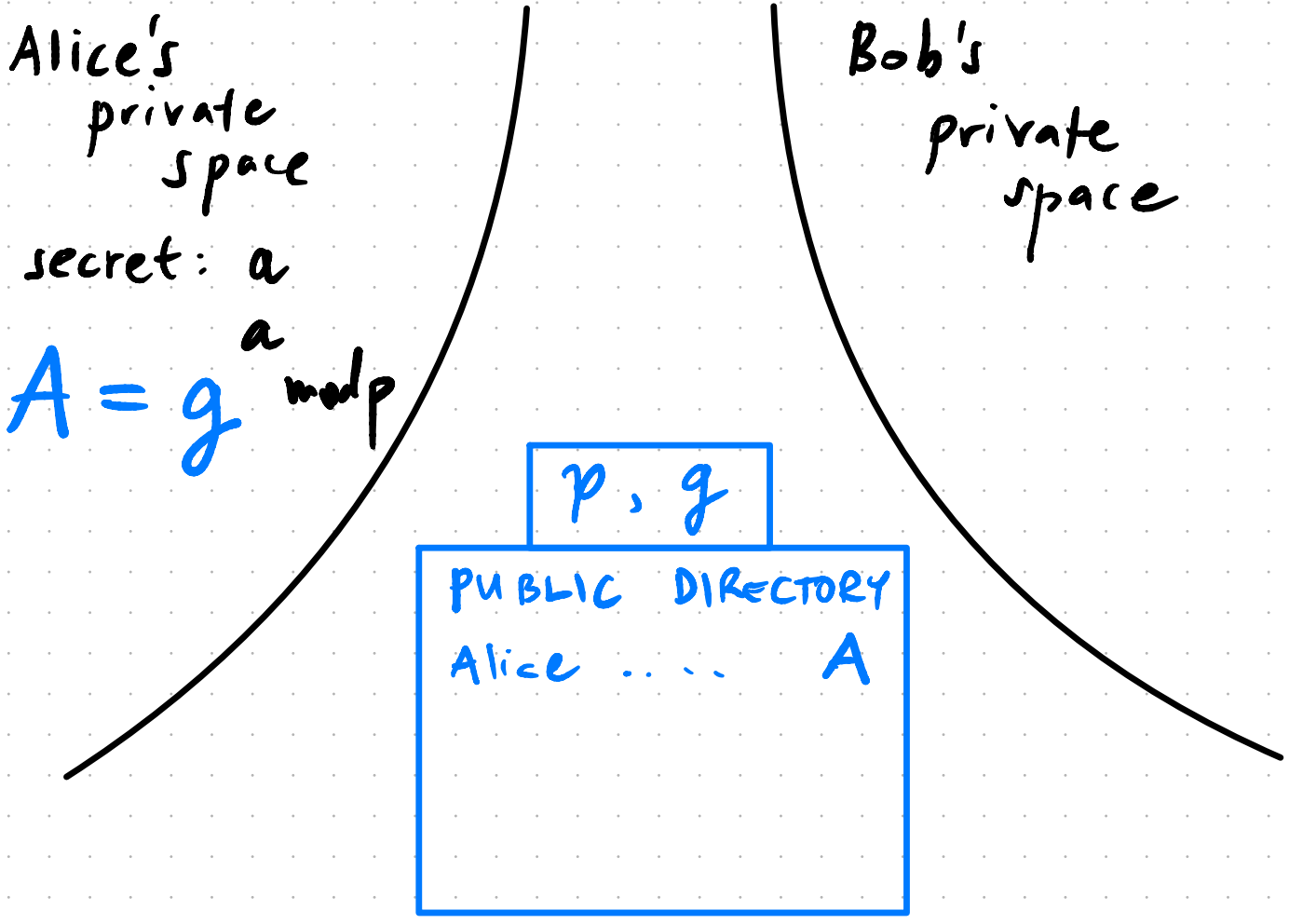
$g = 3$

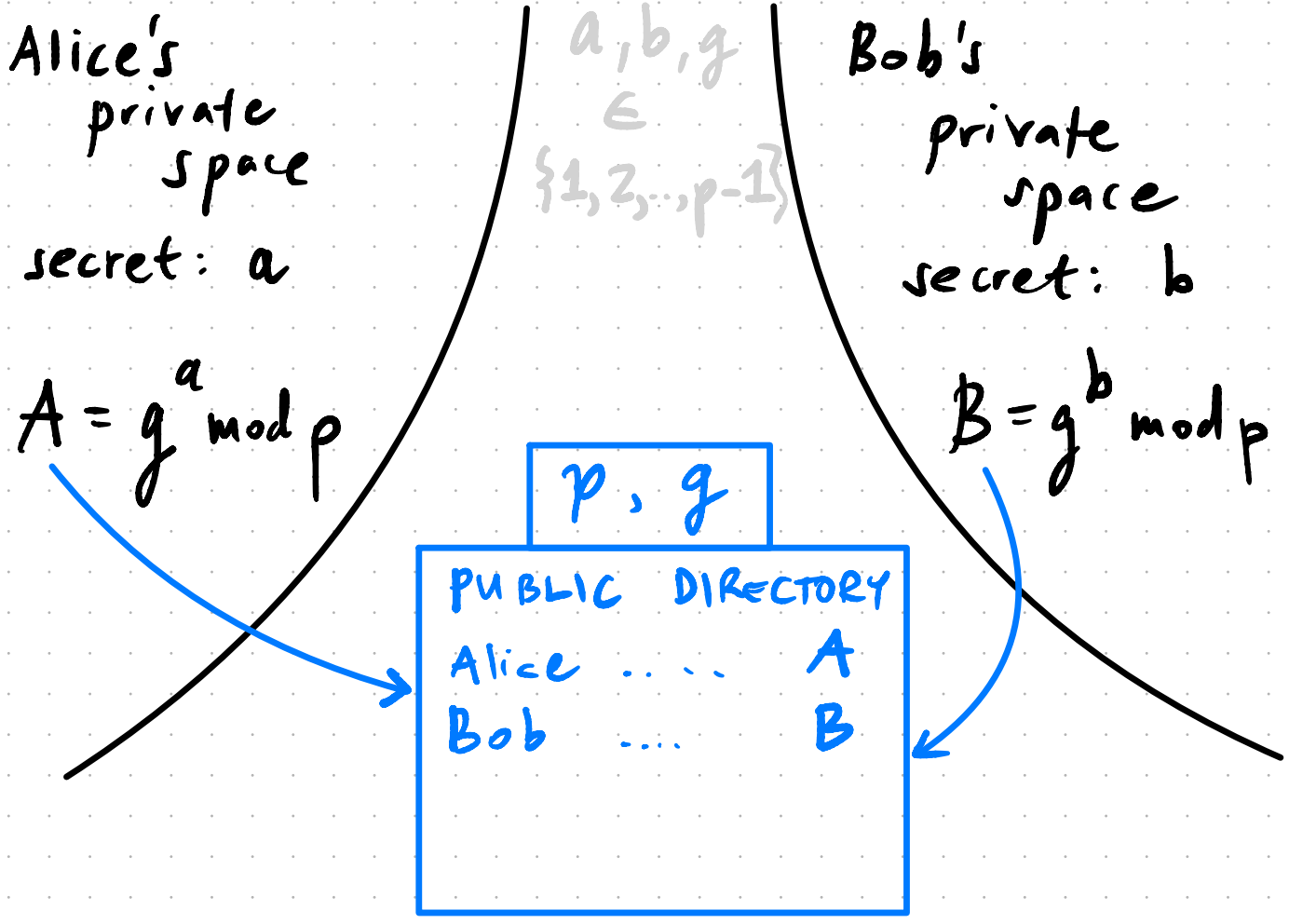
i	$g^i \bmod 7$
0	
1	3
2	2
3	6
4	4
5	5
6	1

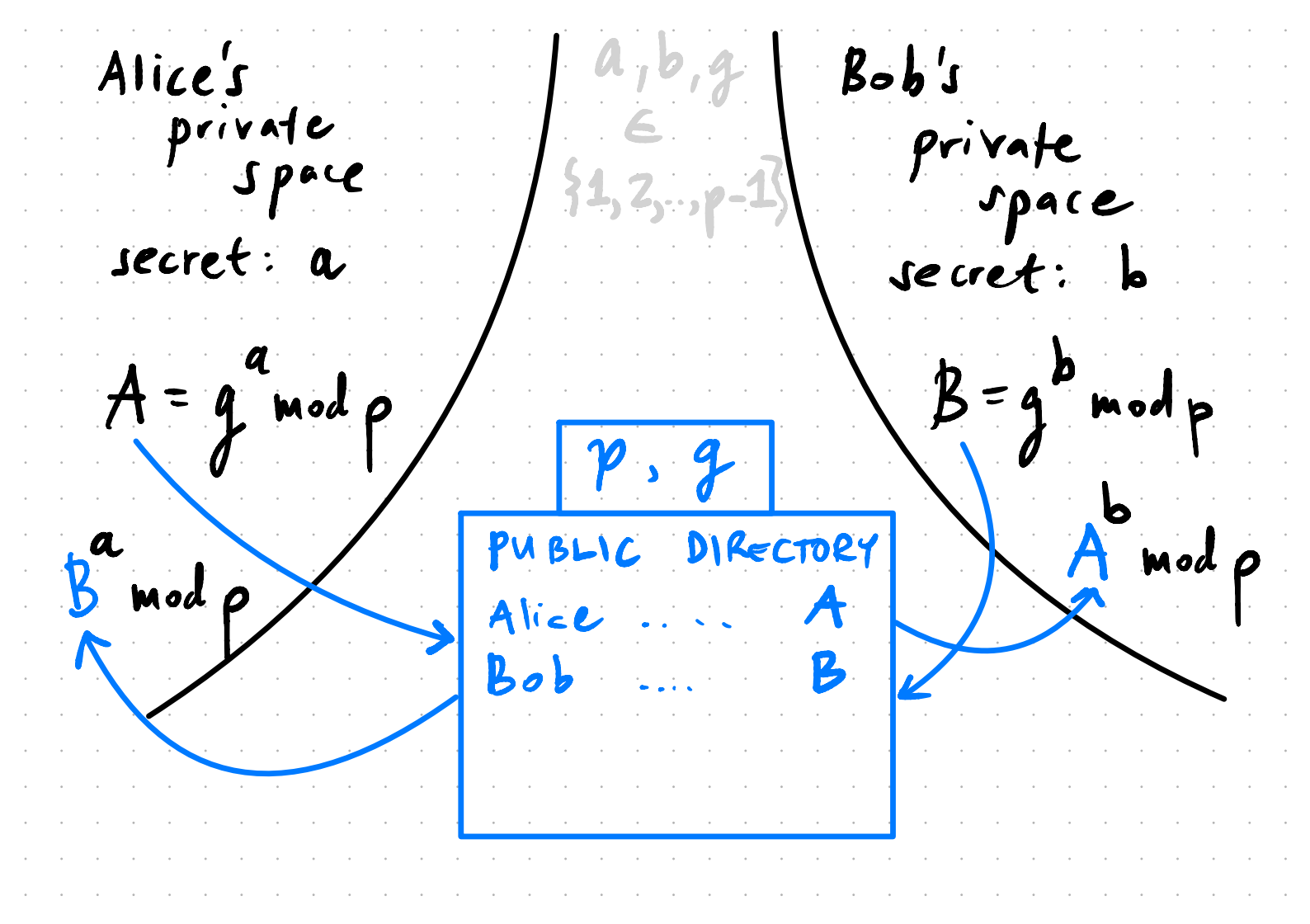
$g = 2$

i	$g^i \bmod 7$
0	
1	2
2	4
3	1
4	2
5	4
6	1


Alice's private space
plaintext
t
$t_1, t_2, t_3, \ldots$
ENCRYPT
KEY
k_A
CYPHERTEXT
CRYPTOGRAM
c
Trudy
Bob's private space
t
DECRYPT
KEY
k_B


Alice's private space
secret: a
$A = g^a \bmod p$
Bob's private space
p, g
PUBLIC DIRECTORY
Alice ... A


Alice's private space
secret: a
$A = g^a \bmod p$
a, b, g ∈ {1, 2, ..., p-1}
Bob's private space
secret: b
$B = g^b \bmod p$
p, g
PUBLIC DIRECTORY
Alice .... A
Bob .... B


Alice's private space
secret: a
$A = g^a \bmod p$
$a, b, g \in \{1, 2, \ldots, p-1\}$
Bob's private space
secret: b
$B = g^b \bmod p$
p, g
PUBLIC DIRECTORY
Alice .... A
Bob .... B
$B^a \bmod p$
$A^b \bmod p$

AT THIS POINT:

- ALICE HAS $B^a \bmod p$

$$B^a = (g^b \bmod p)^a \bmod p$$

- BOB HAS $A^b \bmod p$

$$A^b = (g^a \bmod p)^b \bmod p$$

FACT: [sneak preview of next week...]

For all $x, y, m \in \mathbb{Z}$

$$\left[(x \bmod m) \cdot (y \bmod m)\right] \bmod m$$

$$\equiv xy \pmod m$$

AT THIS POINT:

- ALICE HAS $B^a \bmod p$

$$B^a = (g^b \bmod p)^a \bmod p = g^{ab} \bmod p$$

- BOB HAS $A^b \bmod p$

$$A^b = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p$$

BUT CAN'T ANYBODY GENERATE THIS KEY ?

⟶ NEED TO PERFORM INVERSE OF $g^i \bmod p$.

"DISCRETE LOGARITHM" PROBLEM

Setup:

- Fix a large prime number p. Hereafter all the numbers are in $\{0, 1, \ldots, p-1\}$ and arithmetic is modulo p (more on it later).
- Pick a generator g. A generator has the property that g^i generates all elements in $\{1, 2, \ldots, p-1\}$ when $i = 0, 1, \ldots, p-2$.
- *Note:* Towards the end of this chapter, after introducing all of the algebra necessary, we will see that a generator always exists since we are in what is called a *cyclic group.*

Example

$p = 5$. The numbers are $\{0, 1, 2, 3, 4\}$.

$g = 2$ is a generator. Indeed:

i	g^i
0	1
1	2
2	4
3	3

- Alice picks a number a, kept secret.
- Bob picks a number b, kept secret.
- Alice and Bob send the number $A = g^a$ and $B = g^b$ to the public directory, respectively. This can be done over a non-private channel.

The public directory, readable by everyone, looks like this

User	Public Key
Alice	A
Bob	B
$\vdots$	$\vdots$

Shared key generation:

Suppose that Alice and Bob want to communicate using a symmetric-key cryptosystem (the only kind of cryptosystem that we have studied so far).

They need a shared key k that nobody else knows.

Here is how they proceed:

- Alice gets B from the public directory and computes $k = B^a = g^{ba}$.
- Bob gets A from the public directory and computes $k = A^b = g^{ab}$.

We see that Alice and Bob have come up with a shared key k.

Eve wants to listen in:

Assuming that the cryptosystem used by Bob and Alice is secure, the best option for Eve is to find the key k.

She knows p, g, A, and B.

In general, there seems to be no better way than finding the number a for which $g^a = A$, and then compute $k = B^a$.

This is a problem. Let us check out some numbers: Suppose p is a 2048-bit number. (It must be prime, but let us neglect this and assume $p = 2^{2048}$.)

- It takes roughly
$$2 \log_2 p = 4096$$
multiplications to perform $a \to g^a$ (called discrete exponentiation). With a computer that performs 10^{10} multiplications per second, the exponentiation is done seamlessly.

- It takes roughly
$$\exp\left(\left(\frac{64}{9}\right)^{\frac{1}{3}} (\ln p)^{\frac{1}{3}} \ln\ln p)^{\frac{2}{3}}\right) \approx 10^{35}$$
multiplications to perform $g^a \to a$ (called discrete logarithm to the base g). With the same computer, it takes about 10^{25} seconds, which is about 7×10^7 times the age of the Earth. (The age of the Earth is about 4.5×10^9 years, i.e., 14.3×10^{16} seconds.)

Conclusion: Diffie and Hellman's public key-distribution scheme is clever, efficient, and it seems to be secure.

A Paradigm Shift

The perceived security of the DH public key-distribution algorithm relies on the solution to a problem considered to be difficult to solve.

We call this computational security. Even though it seems unlikely, someone could find a very fast algorithm to compute the discrete logarithm. The DH system would instantly become insecure.

To the contrast, perfect secrecy offers provable security even when the enemy has infinite time and computing power.

Most cryptographic systems rely on computational security.

This leads to the notion of a **one-way function**.

Discrete exponentiation is an example of a one-way function: a function for which a fast algorithm exists and no fast algorithm is known for the function's inverse.

(More precisely, to be considered as a one-way function, the modulus p needs to be a large prime number such that $n = p - 1$ has a large prime factor.)

In the DH protocol:

- Alice uses the function $f_a : g \mapsto g^a$ (with a kept secret)
- Bob uses the function $f_b : g \mapsto g^b$ (with b kept secret)

The functions commute: $f_a(f_b(g)) = f_b(f_a(g))$. Hence

$$f_a(B) = f_b(A).$$

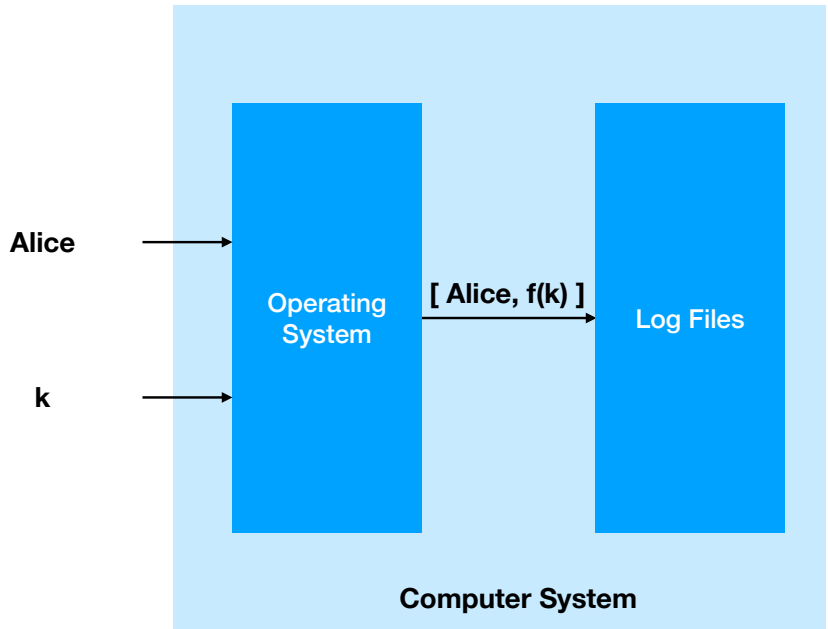
An attacker needs to invert the map $a \mapsto g^a$ (or, equivalently, invert the map $b \mapsto g^b$). This is hard to do, because discrete exponentiation is believed to be a one-way function.

The following is another application of a one-way function.

Example (Application of a one-way function)

If a computer were to save user's names and passwords, a system manager would have access to both.

This is not the case if the operating system stores, along the name, a one-way function f of our password. (The password itself is never stored.)



Trapdoor One-Way Functions

A trapdoor one-way function is a one-way function with an extra feature called the trapdoor information: with this information, the hard-to-carry-out inverse computation becomes easy.

Diffie and Hellmann realized that with such a tool the key-distribution problem would disappear.

Let us first take a look at what Diffie and Hellman proposed to do with a trapdoor one-way function (if such a function could be found).

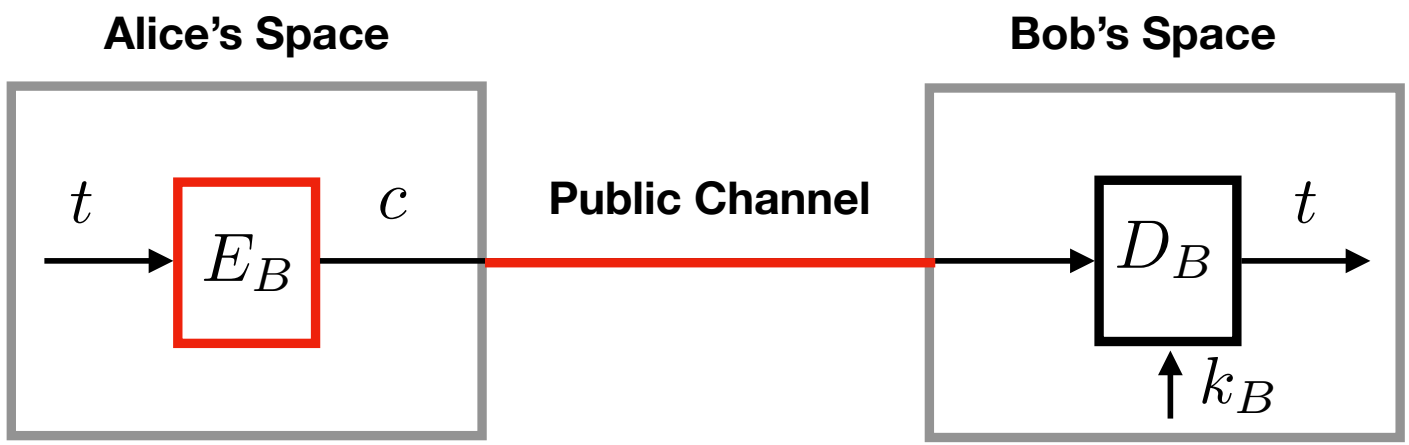
PUBLIC-KEY CRYPTOGRAPHY (ASYMMETRIC CRYPTOGRAPHY)

Suppose that Alice wants to send private information to Bob.

Bob has a trapdoor one-way function, implemented by an algorithm E_B that he publishes in a open directory.

He is the only one who has the trapdoor information k_B. Hence he has the algorithm D_B that implements the inverse function.

Alice and Bob no longer need a shared key (see figure).



The first trapdoor one-way function was found by Rivest, Shamir, and Adleman in 1978.

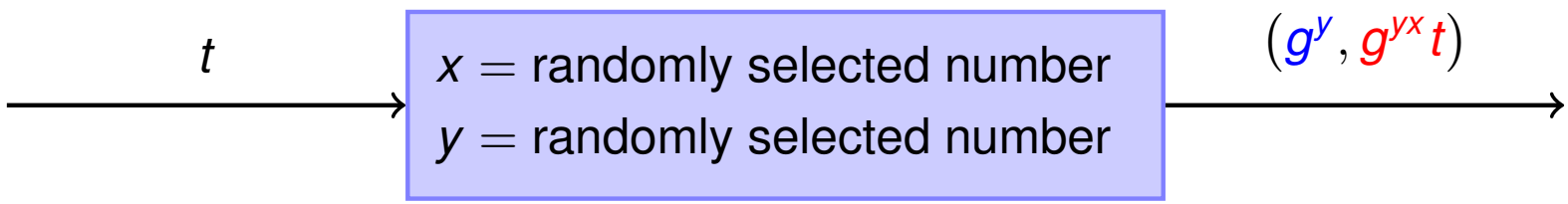
We start with ElGamal's trapdoor one-way function because, as for the DH protocol, it relies only on the assumption that the exponentiation is a one-way function.

ElGamal's trapdoor one-way function

Setup: (same as for the DH protocol)

- Fix a large prime number p. Hereafter all the numbers are in $\{0, 1, \ldots, p-1\}$ and arithmetic is modulo p (more on it later).
- Pick a generator g.
- Pick randomly selected numbers x and y. Unlike p and g, x and y are kept secret.

Here is a trapdoor one-way function, with **trapdoor information** x.



Given the trapdoor information x, we can invert the function as follows:

- compute the inverse of $\left(g^y\right)^x = g^{yx}$.
- multiply the result with $g^{yx}t$. The result is t.

ElGamal's Encryption Scheme

It is based on the above trapdoor one-way function. Let p and g be fixed and known to everyone.

Here is how Alice sends encrypted text to Bob:

Alice:
$t =$ plaintext, t;
$y =$ random number, y;

Bob:
$x =$ random number, x;

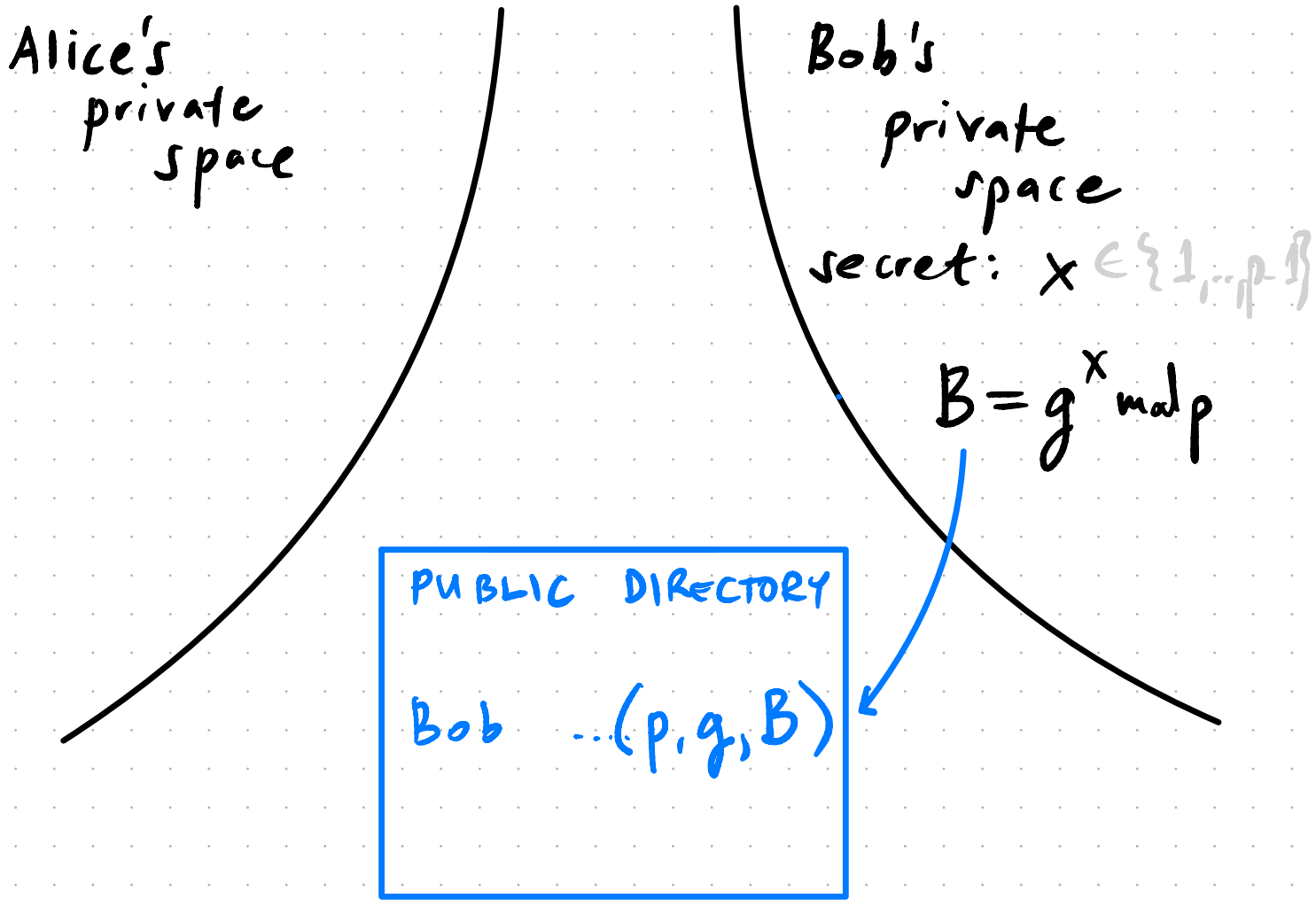
Bob sends g^x to Alice

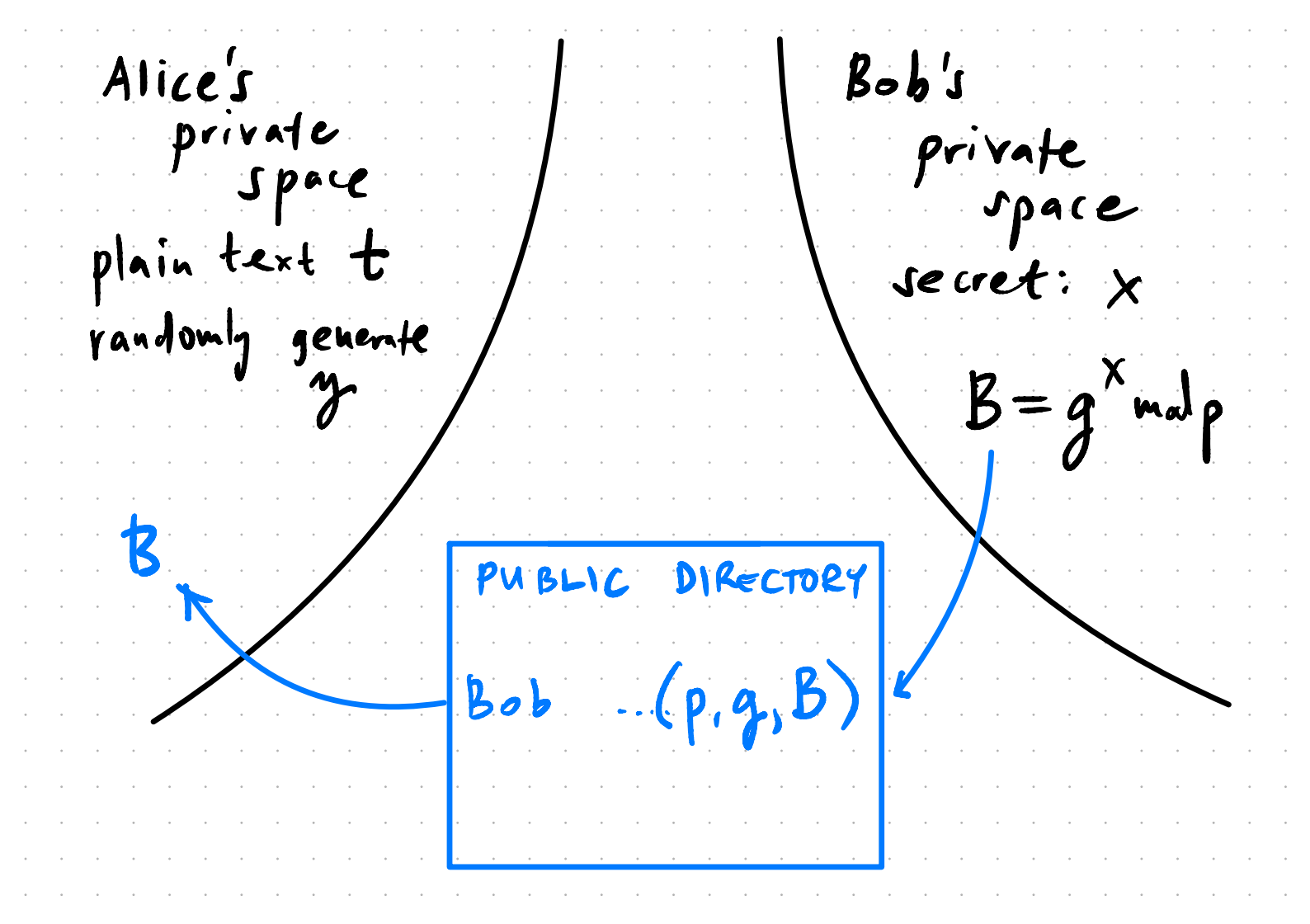
$\longleftarrow$

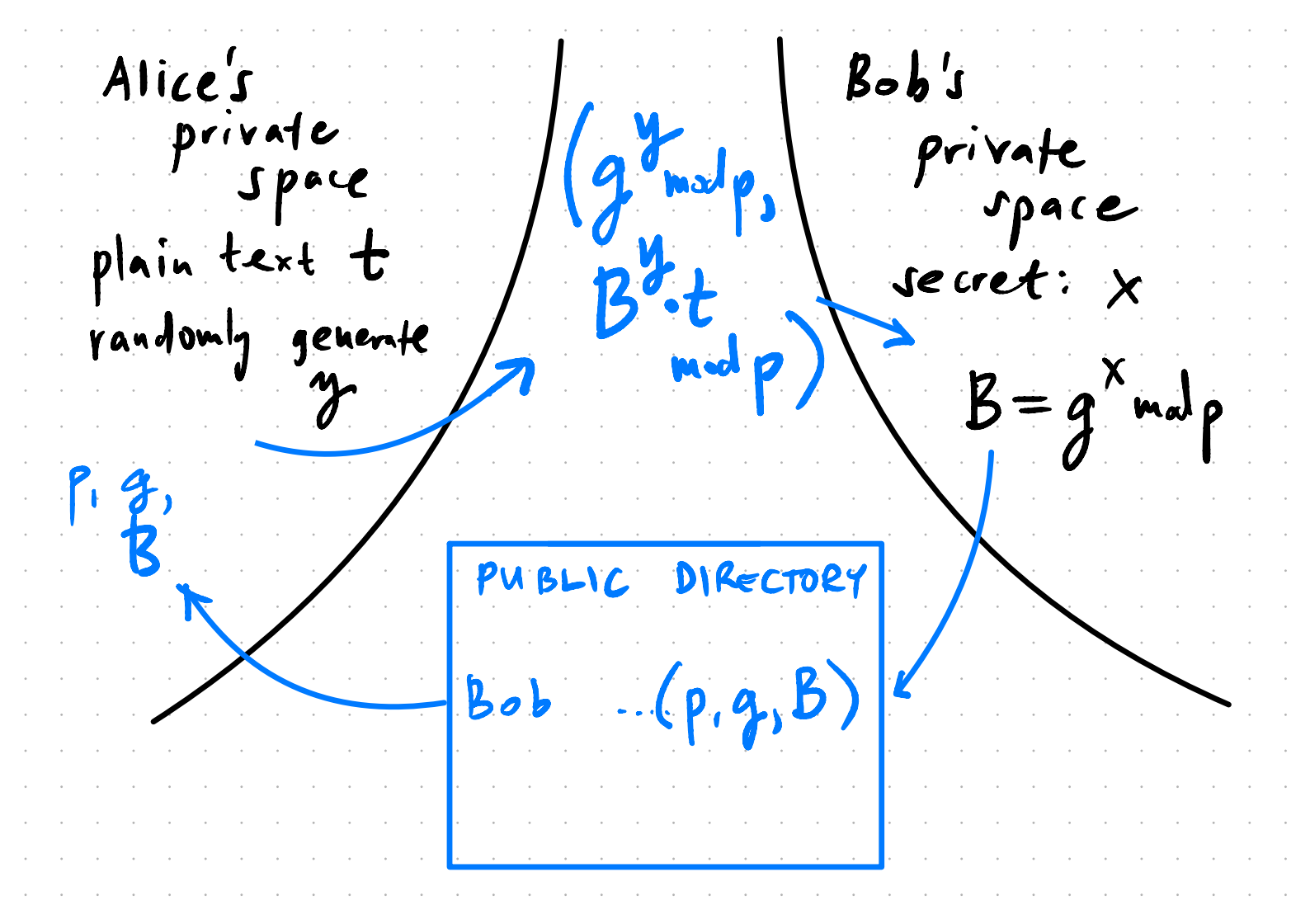
Alice sends the cryptogram $(g^y, g^{yx}t)$ to Bob

$\longrightarrow$

Note: x and y are transaction specific.


Alice's private space
Bob's private space
secret: $x \in \{1, \ldots p-1\}$
$B = g^x \bmod p$
PUBLIC DIRECTORY
Bob ... (p, g, B)


Alice's private space
plain text t
randomly generate y
Bob's private space
secret: x
$B = g^x \bmod p$
B
PUBLIC DIRECTORY
Bob ... (p, g, B)


Alice's private space
plain text t
randomly generate y
Bob's private space
secret: x
$B = g^x \bmod p$
$(g^y \bmod p,\ B^y \cdot t \bmod p)$
p, g, B
PUBLIC DIRECTORY
Bob ... (p, g, B)

IS THIS GOOD ENOUGH FOR BOB?

BOB HAS

x

$g^y \bmod p$

$B^y \cdot t \bmod p$

① $\left(g^y \bmod p\right)^x \bmod p$

$= g^{xy} \bmod p$

$= \left(g^x \bmod p\right)^y \bmod p$

$= B^y \bmod p$

IS THIS GOOD ENOUGH FOR BOB?

BOB HAS

x

$g^y \bmod p$

$B^y \cdot t \bmod p$

① $(g^y \bmod p)^x \bmod p$

$= g^{xy} \bmod p$

$= B^y \bmod p$

② "sneak peek"

There exists a $C \in \{1, \ldots p-1\}$ such that $(C \cdot B^y) \bmod p = 1$

such a C is called

"MULTIPLICATIVE INVERSE OF B^y"

It is unique.

Ex. $p = 7$.

a	Multiplicative inverse of a
1	1
2	4
3	5
4	2
5	3
6	6

such a C is called

"MULTIPLICATIVE INVERSE OF B^y"

It is <u>unique</u>.

<u>Ex.</u> $m = 6$

a	Multiplicative inverse of a
1	1
2	———
3	
4	
5	

Step ②, continued:

Compute C and find:

$$\{C \cdot [(B^y \cdot t) \bmod p]\} \bmod p$$

$$= C \cdot B^y \cdot t \bmod p$$

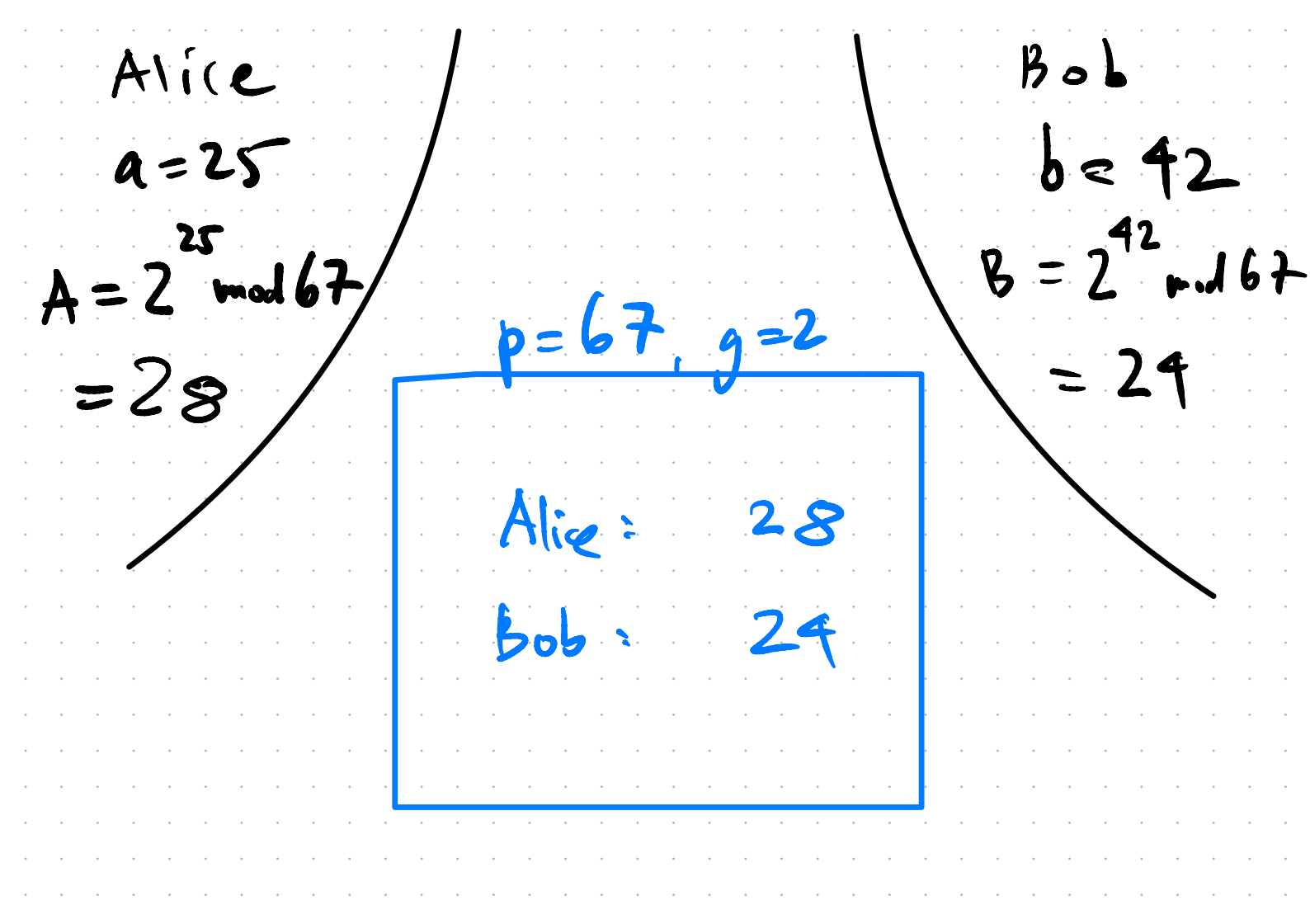
$$= \{([C \cdot B^y] \bmod p) \cdot t\} \bmod p$$

$$= t \bmod p = t$$

BURNING QUESTIONS

- "modulo": how does this work, really
- why did Diffie-Hellman pick a prime number p?
- which g work? (what is the structure behind it?)

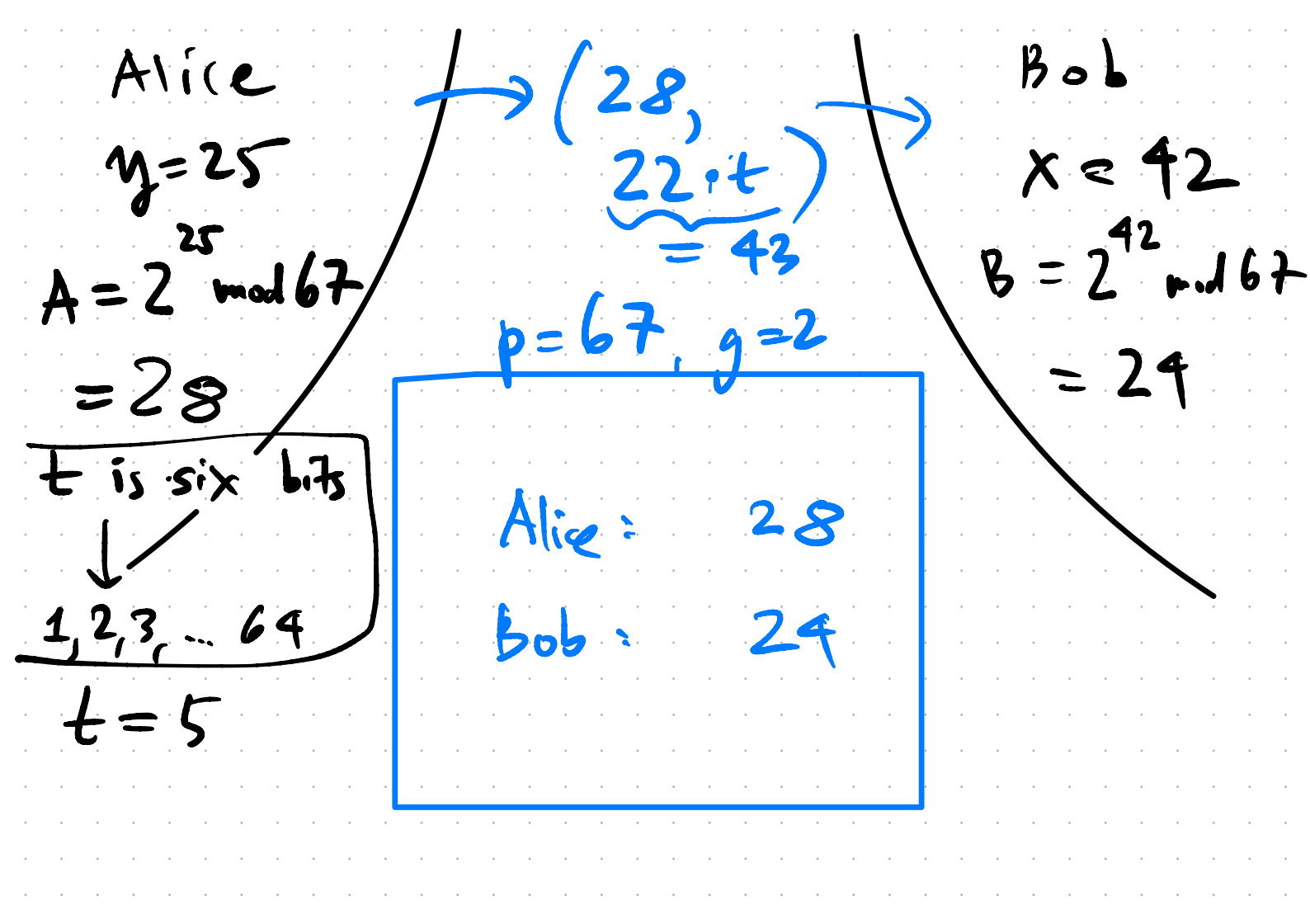
- what about "multiplicative inverses"?


Alice
a = 25
A = 2^25 mod 67
= 28
Bob
b = 42
B = 2^42 mod 67
= 24
p = 67, g = 2
Alice : 28
Bob : 24

SHARED SECRET:

Alice computes: $29^{25} \bmod 67 = 22$

Bob computes: $28^{92} \bmod 67 = 22$


Alice
y = 25
A = 2^25 mod 67
= 28
t is six bits
1, 2, 3, ... 64
t = 5
(28, 22, t)
= 43
p = 67, g = 2
Alice: 28
Bob: 24
Bob
x = 42
B = 2^42 mod 67
= 24

BOB RECEIVES $(28, 43)$.

① HE CALCULATES:

$$28^{92} \bmod 67 = 22$$

② FIND MULTIPLICATIVE INVERSE OF 22 $(\bmod 67)$:

$$64 \cdot 22 \bmod 67 = 1$$

$\Rightarrow$ WE HAVE $d = 64$

FINALLY, HE CALCULATES:

$$64 \cdot 43 \mod 67$$

$$= 5$$

↳ HE NOW KNOWS $\underline{t = 5}$.

Next goal: The RSA public key cryptosystem. It will take two weeks to build up the necessary background.