ARIEL UNIVERSITY

אוניברסיטת אריאל בשומרון

Foundations of Modern Cryptography

Edward A. Hirsch

https://edwardahirsch.github.io/edwardahirsch

Neapolis University Pafos

Lecture 1: October 7, 2024

Setting the stage

Schemes, Protocols and Primitives

Alice

Bob

Setting the stage

Schemes, Protocols and Primitives

Alice

Bob

Setting the stage

Schemes, Protocols and Primitives

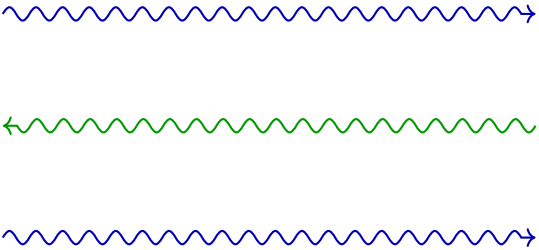
Alice

Bob

Setting the stage

Schemes, Protocols and Primitives

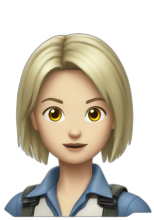
Alice



Bob

Setting the stage

Schemes, Protocols and Primitives



Alice

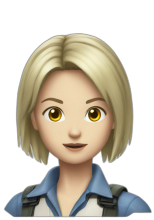
Bob

Eve

Pictures courtesy AI Emojis Generator

Setting the stage

Schemes, Protocols and Primitives



Alice

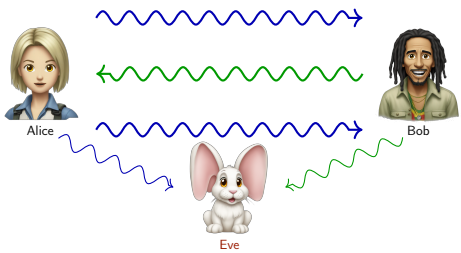
Bob

Charlie

Pictures courtesy AI Emojis Generator

Setting the stage

Schemes, Protocols and Primitives



`Cryptographic Task:`

- Honest and dishonest participants
- Goal (send a message, compute a function, . . .)
- Desired security (cannot fake a message, cannot learn a key, . . .)

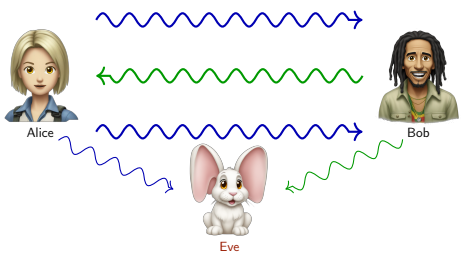
`Cryptographic Scheme:`

- How to perform this Task using “primitives”.

Pictures courtesy AI Emojis Generator

Setting the stage

Schemes, Protocols and Primitives



`Cryptographic Primitive:`

- A "building" block of protocol
- Abstract notion satisfying certain properties
  - Something easy to compute
  - Something hard to compute

`Cryptographic Task:`

- Honest and dishonest participants
- Goal (send a message, compute a function, ...)
- Desired security (cannot fake a message, cannot learn a key, ...)

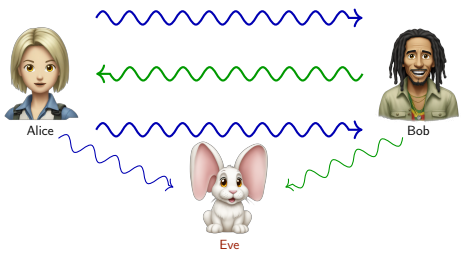
`Cryptographic Scheme:`

- How to perform this Task using "primitives".

Pictures courtesy AI Emojis Generator

Setting the stage

Schemes, Protocols and Primitives



`Cryptographic Task:`

- Honest and dishonest participants
- Goal (send a message, compute a function, . . .)
- Desired security (cannot fake a message, cannot learn a key, . . .)

`Cryptographic Scheme:`

- How to perform this Task using "primitives".

`Cryptographic Primitive:`

- A "building" block of protocol
- Abstract notion satisfying certain properties
  - Something easy to compute
  - Something hard to compute

`Implementation:`

- Specific implementation of primitives (e.g., the trapdoor RSA function)
- Thus specific protocol

Pictures courtesy AI Emojis Generator

Course Overview

- Cryptographic Primitives
- Cryptographic Schemes
- Security Definitions
- Examples

- One-way functions
- Trapdoor functions
- Oblivious transfer
- Cryptographic hash-functions
- Security basics: indistinguishability and semantic security
- Public-key encryption
- Key agreement
- Private-key encryption
- Digital signatures
- Bit commitment (bid in a sealed envelope)
- Secure distributed function evaluation
- Blockchains

Lecture Plan

- Appetizer: Main Cryptographic Tasks
- Background Material
- Is Cryptography Possible?

Task: Public-Key Encryption

Alice
public key, secret key

Eve

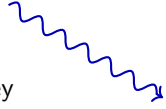
Bob
message

Task: Public-Key Encryption

public key

Alice

public key, secret key



Eve

Bob

message

Task: Public-Key Encryption

Alice
public key, secret key

public key

encrypted message

Eve

Bob
message

Task: Public-Key Encryption

Alice
public key, secret key

public key

encrypted message

Bob
message

Eve

Eve should be unable to decrypt (not to say recover the secret key)
even though she has seen a lot of communication (and perhaps some messages).

Task: Private-Key Encryption

Alice
secret key

Eve

Bob
message, secret key

Task: Private-Key Encryption

Alice
secret key

encrypted message

Bob
message, secret key

Eve

Task: Private-Key Encryption

Alice
secret key

encrypted message

Bob
message, secret key

Eve

Eve should be unable to decrypt (not to say recover the secret key)
even though she has seen a lot of communication (and perhaps some messages).

Task: Private-Key Encryption

Alice
secret key

encrypted message

Bob
message, secret key

Eve

Eve should be unable to decrypt (not to say recover the secret key)
even though she has seen a lot of communication (and perhaps some messages).

There are lots of messages! The total length is much larger than the secret key!

Task: Digital Signatures

Alice

public key, secret key

Bob

—

Charlie

Task: Digital Signatures

Alice

public key, secret key

public key

Bob

—

Charlie

Task: Digital Signatures

Alice

public key, secret key

public key

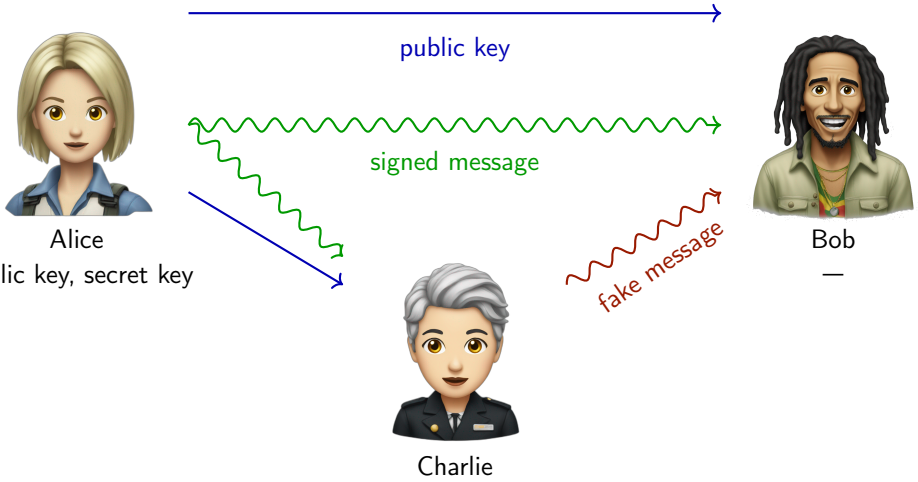
signed message

Bob

—

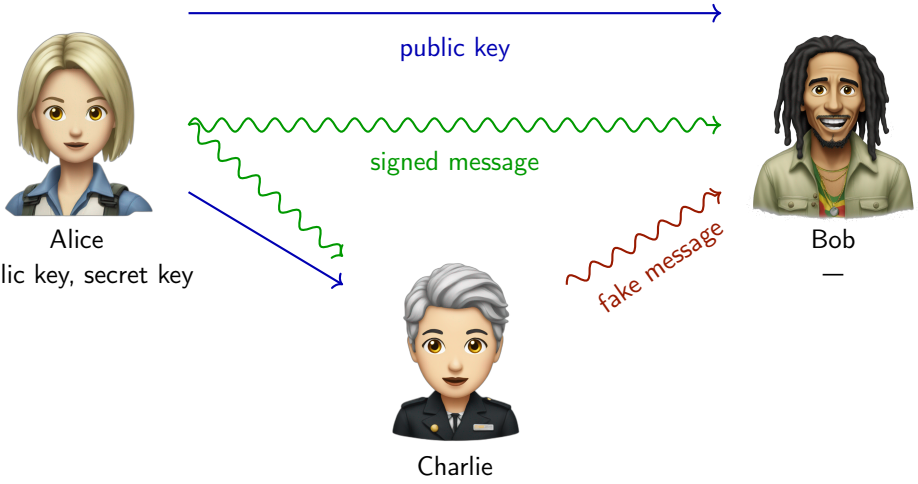
Charlie

Task: Digital Signatures



Charlie should be unable to sign (not to say recover the secret key) even though he has seen a lot of signed messages.

Task: Digital Signatures



Charlie should be unable to sign (not to say recover the secret key) even though he has seen a lot of signed messages.

Perhaps his partner even forced Alice to sign similar messages!

Task: Key Agreement

Alice

Bob

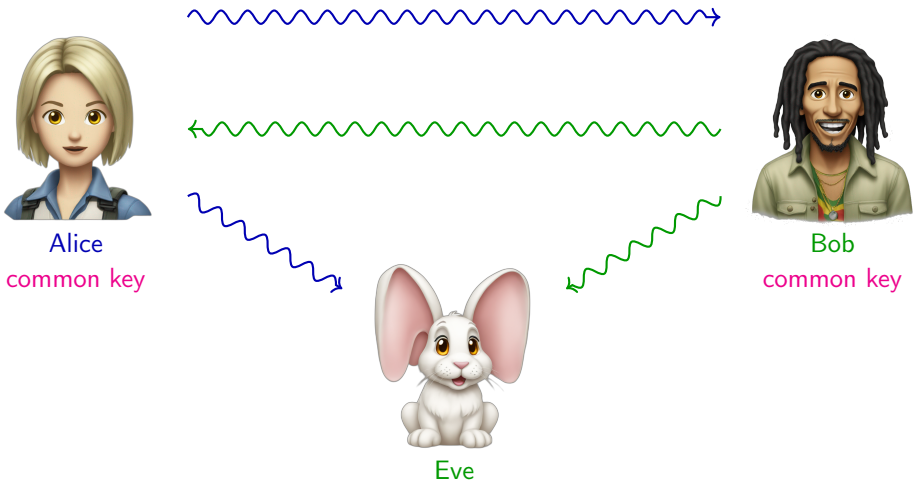
Task: Key Agreement

Alice

Bob

Eve

Task: Key Agreement



The key depends both on Alice and Bob and is hidden from Eve.

Task: Bit (or Bid) Commitment

Alice

bid, key

Public

Task: Bit (or Bid) Commitment

Alice

bid, key

bid

Public

Task: Bit (or Bid) Commitment

Alice

bid, key

bid

key

Public

Alice's bid will be obvious to everybody after (but not before) opening the "envelope".

She should be unable to change her bid retroactively.

Task: Secure Distributed Function Evaluation

Alice

x

Bob

y

Task: Secure Distributed Function Evaluation

Alice

x

Bob

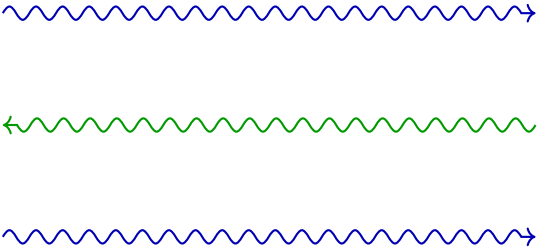
y

Task: Secure Distributed Function Evaluation

Alice

x

$f(x, y)$



Bob

y

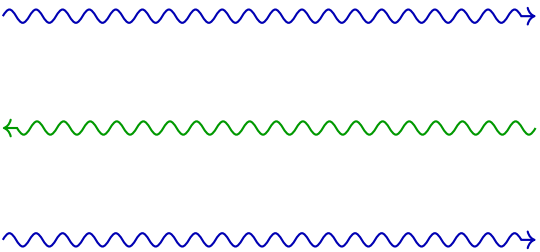
$f(x, y)$

Task: Secure Distributed Function Evaluation

Alice

x

$f(x, y)$



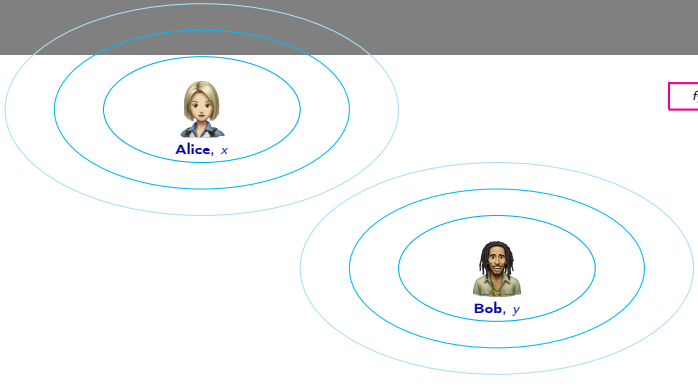
Bob

y

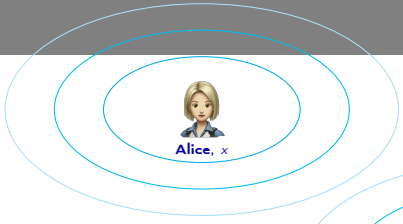
$f(x, y)$

Alice (Bob) does not learn from y (x, resp.) more than needed to compute $f(x, y)$.

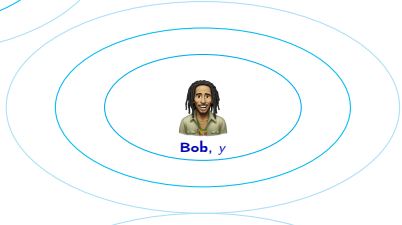
Task: Blockchain



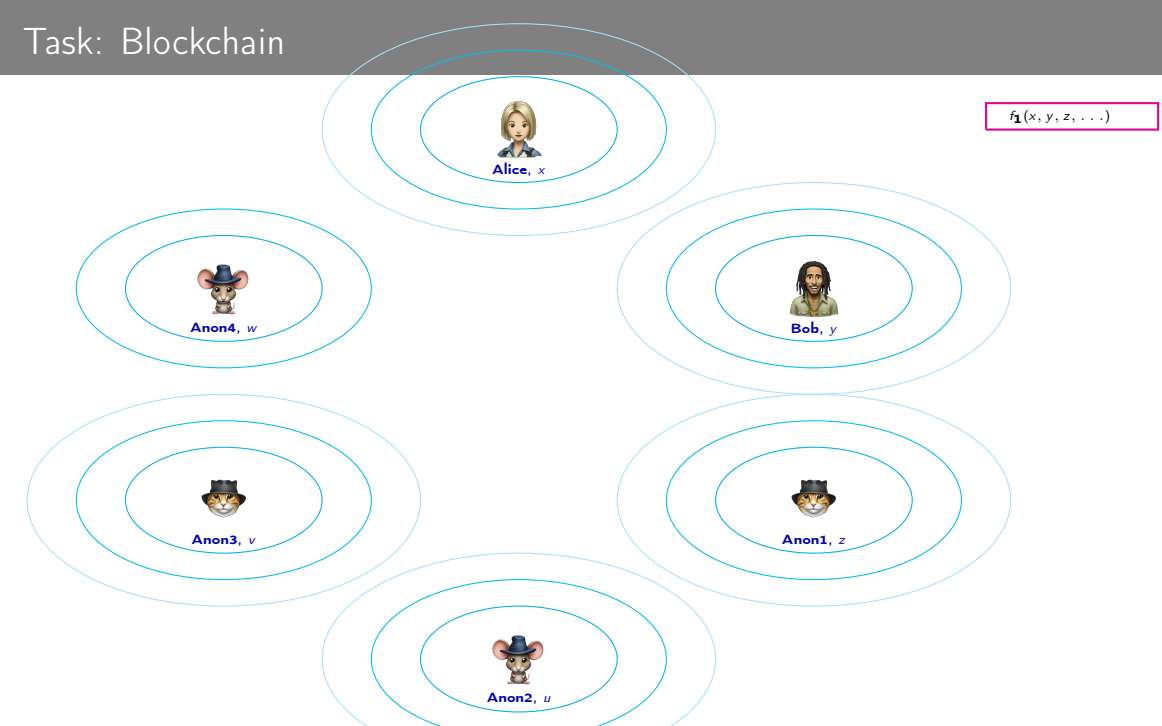
Task: Blockchain



$f_1(x, y, z, \ldots)$

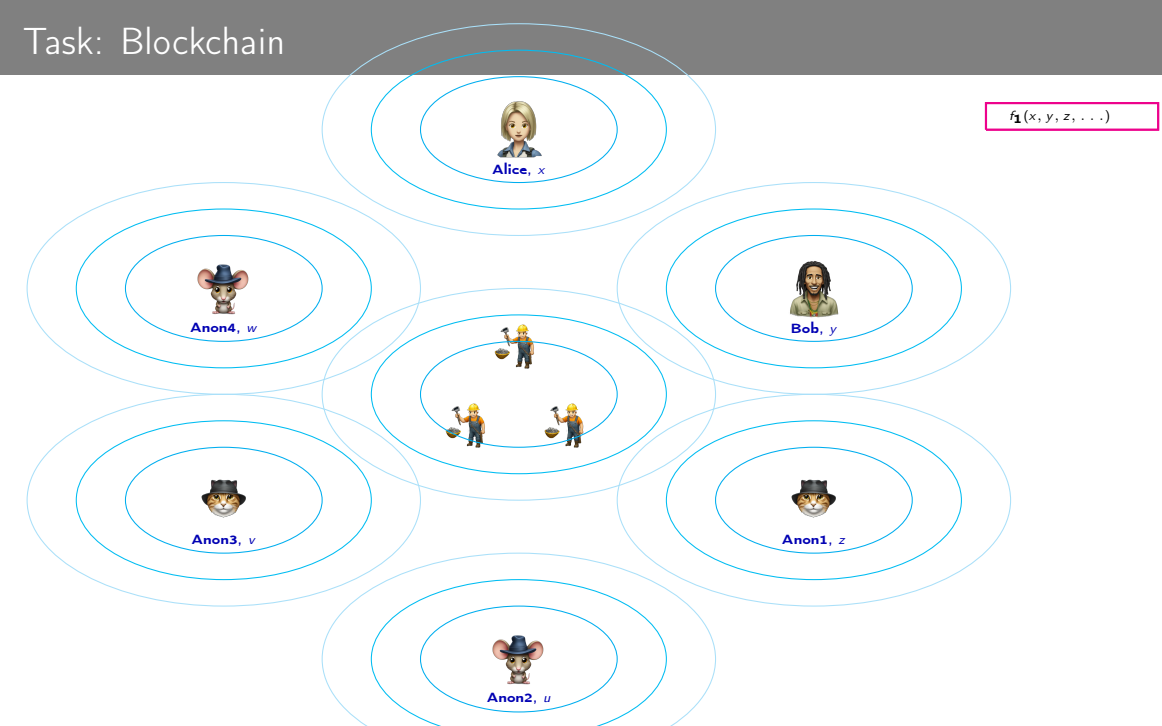


Task: Blockchain

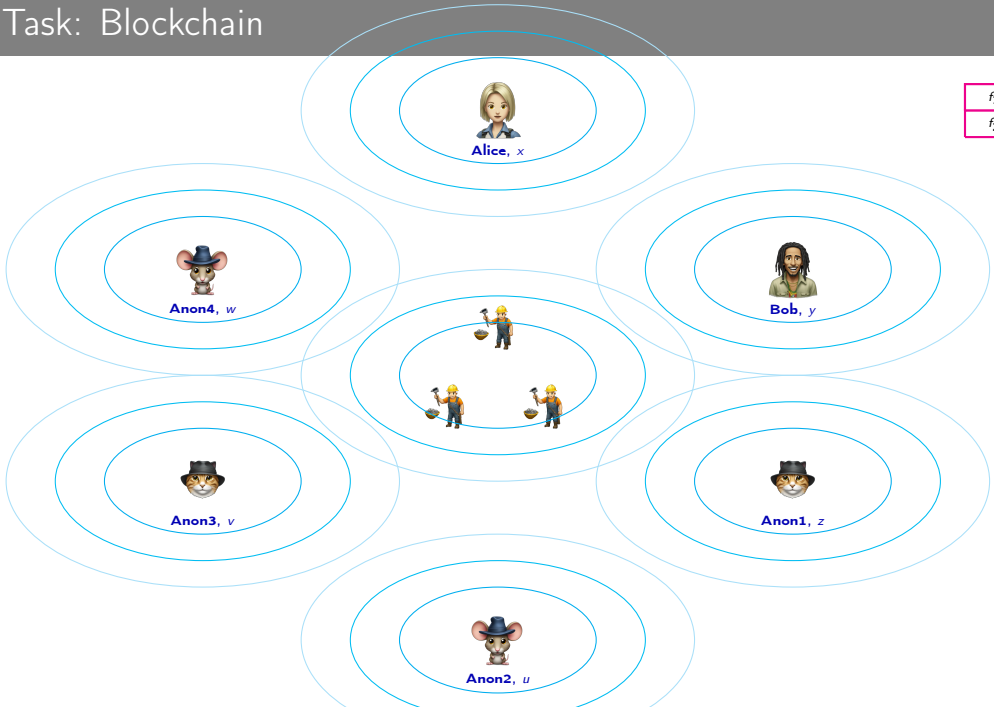


Task: Blockchain

$f_1(x, y, z, \ldots)$



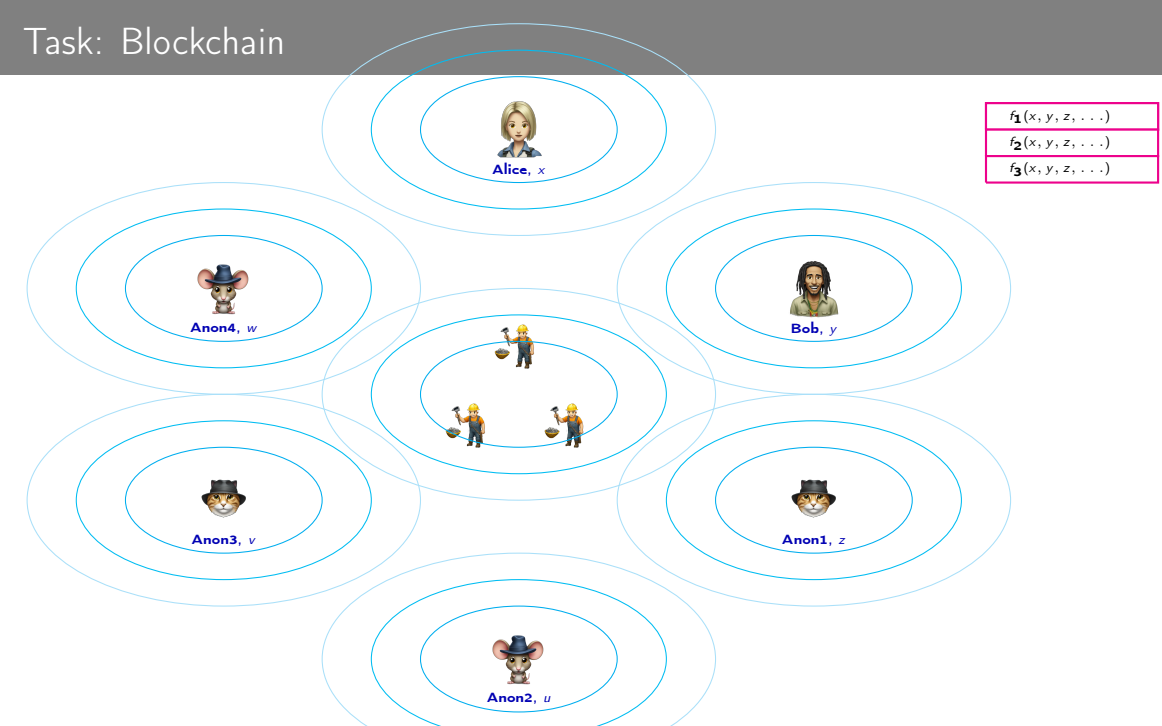
Task: Blockchain



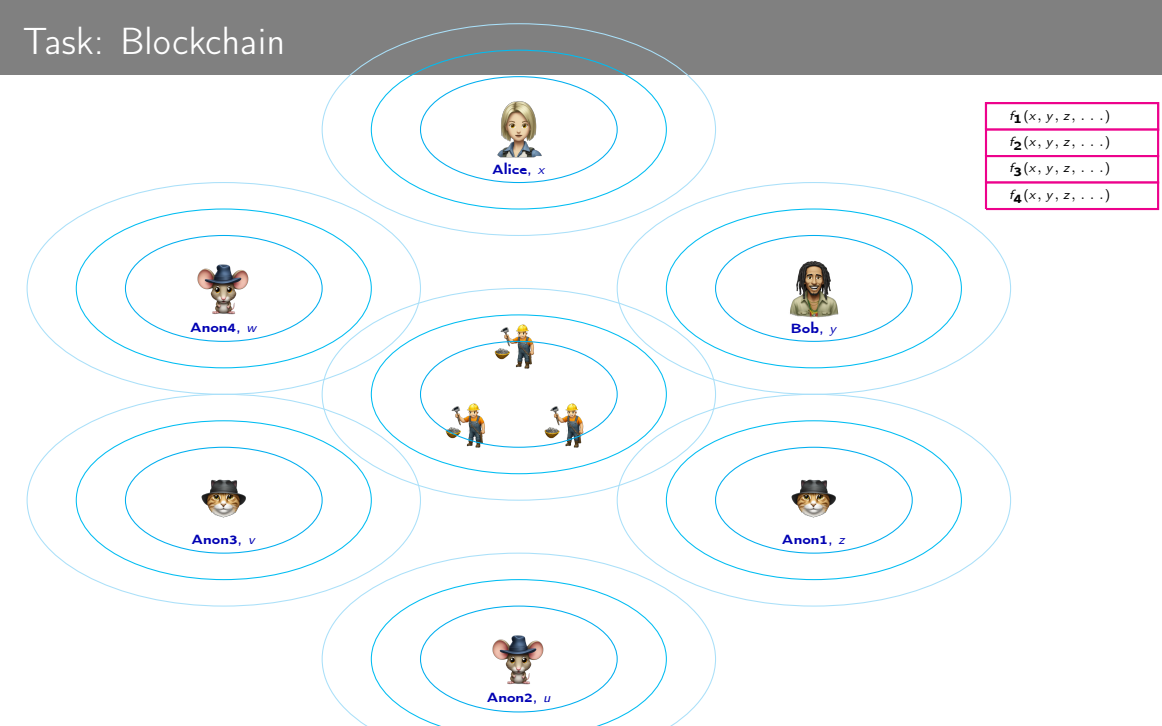
$f_1(x, y, z, \ldots)$

$f_2(x, y, z, \ldots)$

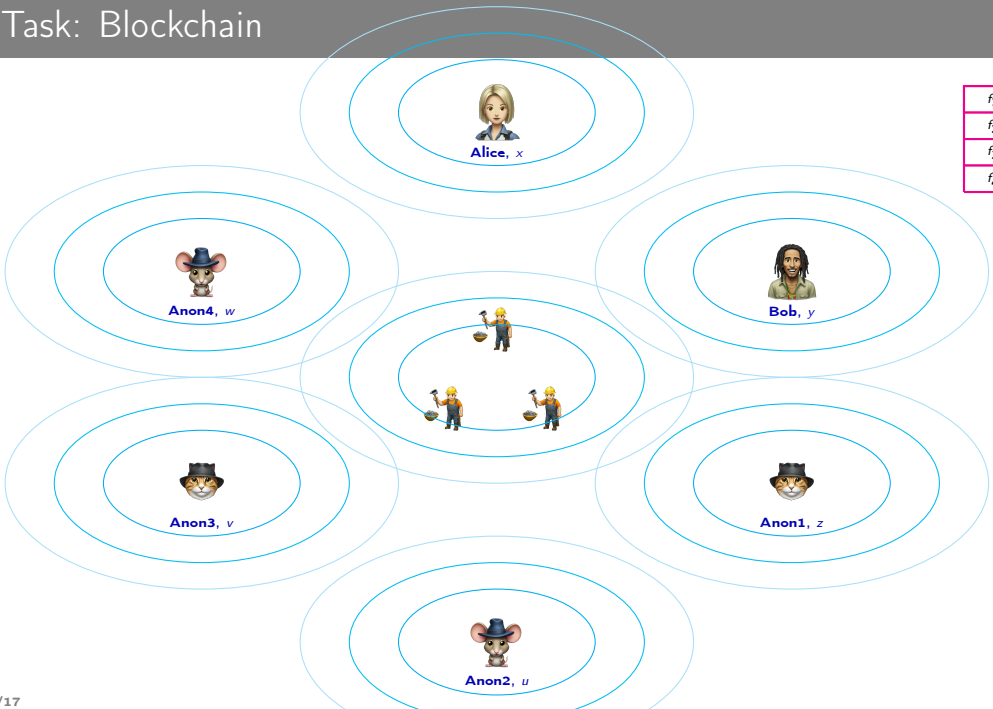
Task: Blockchain



Task: Blockchain



Task: Blockchain



$f_1(x, y, z, \ldots)$

$f_2(x, y, z, \ldots)$

$f_3(x, y, z, \ldots)$

$f_4(x, y, z, \ldots)$

Background 1: **P**, **NP**, and companions

You don't need to remember the notation

P 0/1-questions that can be decided fast (in polynomial time)

Example: does a system of integer linear equations have a solution?[1]

[1] Not absolutely obvious, but true. Simpler examples are equations over $\mathbb{F}_2$ or $2-\mathtt{SAT}$.

[2] A more popular example: $\mathtt{SAT}$.

Background 1: **P**, **NP**, and companions

You don't need to remember the notation

P 0/1-questions that can be decided fast (in polynomial time)
Example: does a system of integer linear equations have a solution?[1]

fP functions that can be computed fast (in polynomial time)
Example: given $n, m \in \mathbb{N}$, compute $n \cdot m$

[1] Not absolutely obvious, but true. Simpler examples are equations over $\mathbb{F}_2$ or $2 - \mathtt{SAT}$.
[2] A more popular example: $\mathtt{SAT}$.

Background 1: **P**, **NP**, and companions

You don't need to remember the notation

P — 0/1-questions that can be decided fast (in polynomial time)
Example: does a system of integer linear equations have a solution?[1]

***f*P** — functions that can be computed fast (in polynomial time)
Example: given $n, m \in \mathbb{N}$, compute $n \cdot m$

NP — 0/1-questions that can be verified fast ***given a solution***
Example: does a system of linear inequalities have a 0/1-solution?[2]

[1] Not absolutely obvious, but true. Simpler examples are equations over $\mathbb{F}_2$ or $2 - \mathtt{SAT}$.
[2] A more popular example: $\mathtt{SAT}$.

Background 1: **P**, **NP**, and companions

You don't need to remember the notation

P	0/1-questions that can be decided fast (in polynomial time) Example: does a system of integer linear equations have a solution?[1]
***f*P**	functions that can be computed fast (in polynomial time) Example: given $n, m \in \mathbb{N}$, compute $n \cdot m$
NP	0/1-questions that can be verified fast ***given a solution*** Example: does a system of linear inequalities have a 0/1-solution?[2]
FNP	search problems corresponding to **NP** Example: provide a 0/1-solution for a system of linear inequalities, if any

[1] Not absolutely obvious, but true. Simpler examples are equations over $\mathbb{F}_2$ or $2 - \mathtt{SAT}$.

[2] A more popular example: $\mathtt{SAT}$.

Background 1: **P**, **NP**, and companions

You don't need to remember the notation

P	0/1-questions that can be decided fast (in polynomial time) Example: does a system of integer linear equations have a solution?[1]
***f*P**	functions that can be computed fast (in polynomial time) Example: given $n, m \in \mathbb{N}$, compute $n \cdot m$
NP	0/1-questions that can be verified fast ***given a solution*** Example: does a system of linear inequalities have a 0/1-solution?[2]
FNP	search problems corresponding to **NP** Example: provide a 0/1-solution for a system of linear inequalities, if any
FP	problems in **FNP** that can be solved fast Example: provide a solution for a system of integer linear equations

We believe in $\mathbf{P} \neq \mathbf{NP}$ (this is equivalent to $\mathbf{FP} \neq \mathbf{FNP}$).
This is required for 99% of cryptography.

[1] Not absolutely obvious, but true. Simpler examples are equations over $\mathbb{F}_2$ or $2 - \mathtt{SAT}$.

[2] A more popular example: $\mathtt{SAT}$.

Background 1: **P**, **NP**, and companions

You don't need to remember the notation

P	0/1-questions that can be decided fast (in polynomial time) Example: does a system of integer linear equations have a solution?[1]
fP	functions that can be computed fast (in polynomial time) Example: given $n, m \in \mathbb{N}$, compute $n \cdot m$
NP	0/1-questions that can be verified fast ***given a solution*** Example: does a system of linear inequalities have a 0/1-solution?[2]
FNP	search problems corresponding to **NP** Example: provide a 0/1-solution for a system of linear inequalities, if any
FP	problems in **FNP** that can be solved fast Example: provide a solution for a system of integer linear equations

We believe in $\mathbf{P} \neq \mathbf{NP}$ (this is equivalent to $\mathbf{FP} \neq \mathbf{FNP}$).
This is required for 99% of cryptography.

Example

Encryption:	$B(\texttt{msg}, \texttt{key}) = \texttt{code}$.	*Should be easy.*	
Illegal decryption:	$E(\texttt{code}) = (\texttt{msg}, \texttt{key})$.	*Should be hard.*	Breaking this is an **FNP** task!

[1] Not absolutely obvious, but true. Simpler examples are equations over $\mathbb{F}_2$ or $2 - \texttt{SAT}$.
[2] A more popular example: SAT.

Background 1: **P**, **NP**, and companions

You don't need to remember the notation

P	0/1-questions that can be decided fast (in polynomial time) Example: does a system of integer linear equations have a solution?[1]
fP	functions that can be computed fast (in polynomial time) Example: given $n, m \in \mathbb{N}$, compute $n \cdot m$
NP	0/1-questions that can be verified fast ***given a solution*** Example: does a system of linear inequalities have a 0/1-solution?[2]
FNP	search problems corresponding to **NP** Example: provide a 0/1-solution for a system of linear inequalities, if any
FP	problems in **FNP** that can be solved fast Example: provide a solution for a system of integer linear equations

We believe in $\mathbf{P} \neq \mathbf{NP}$ (this is equivalent to $\mathbf{FP} \neq \mathbf{FNP}$).
This is required for 99% of cryptography.

Example

Encryption:	$B(\texttt{msg}, \texttt{key}) = \texttt{code}$.	*Should be easy.*	
Illegal decryption:	$E(\texttt{code}) = (\texttt{msg}, \texttt{key})$.	*Should be hard.*	Breaking this is an **FNP** task!

This is by far not enough for cryptography!

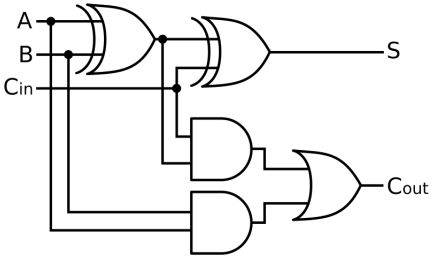
[1] Not absolutely obvious, but true. Simpler examples are equations over $\mathbb{F}_2$ or $2-\texttt{SAT}$.

[2] A more popular example: SAT.

Background 2: Boolean circuits

As viewed by Engineers

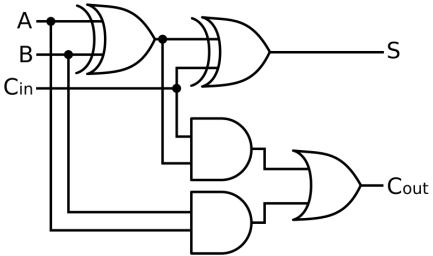
FA (Full Adder)



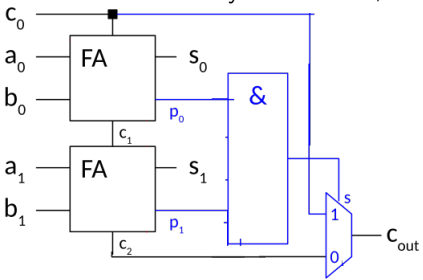
Background 2: Boolean circuits

As viewed by Engineers

FA (Full Adder)



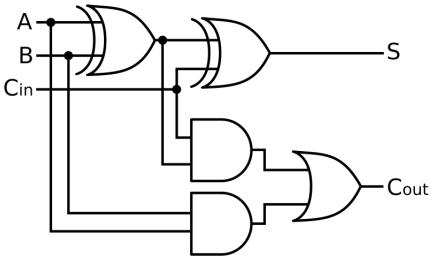
Carry-Save Adder, 2 bits



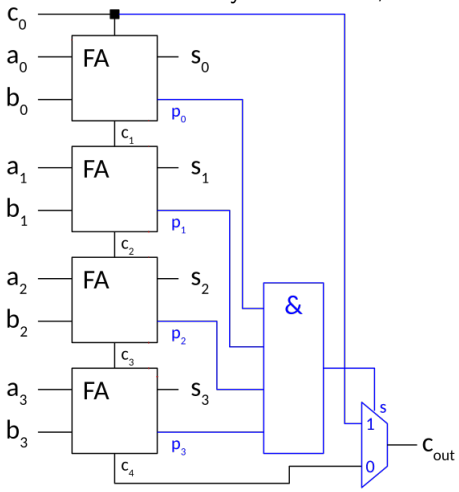
Background 2: Boolean circuits

As viewed by Engineers

FA (Full Adder)



Carry-Save Adder, 4 bits



We need to build many of these if we work with many bits! Think of key length.

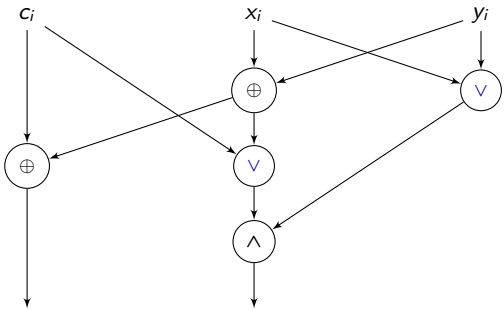
Our adversary does ***not*** need to break the code uniformly on every length.

Background 2: Boolean circuits (non-uniform computations)

As viewed by Computer Scientists

- A directed acyclic graph.
- Binary Boolean operations in the internal nodes ("gates").
- Sources (in-degree zero): inputs, or variables.
- Outputs: designated nodes.

Example: **Full Adder**. Three input bits.

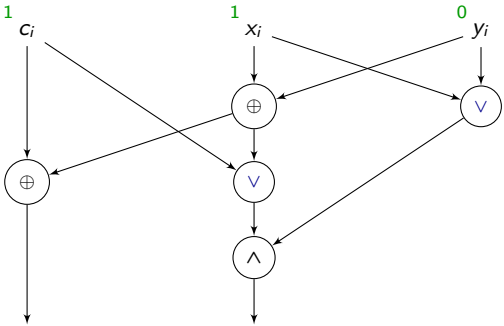


Background 2: Boolean circuits (non-uniform computations)

As viewed by Computer Scientists

- A directed acyclic graph.
- Binary Boolean operations in the internal nodes ("gates").
- Sources (in-degree zero): inputs, or variables.
- Outputs: designated nodes.

Example: **Full Adder**. Three input bits.

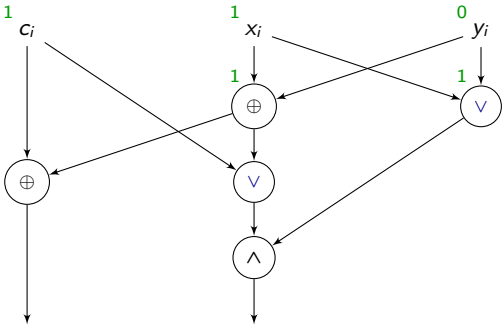


Background 2: Boolean circuits (non-uniform computations)

As viewed by Computer Scientists

- A directed acyclic graph.
- Binary Boolean operations in the internal nodes ("gates").
- Sources (in-degree zero): inputs, or variables.
- Outputs: designated nodes.

Example: **Full Adder**. Three input bits.

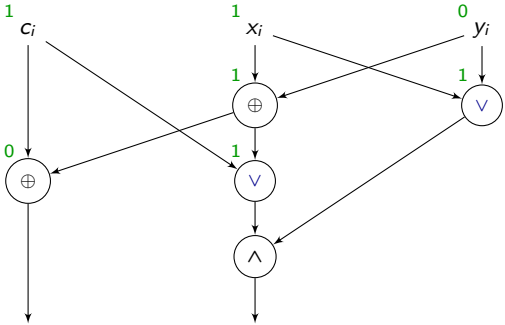


Background 2: Boolean circuits (non-uniform computations)

As viewed by Computer Scientists

- A directed acyclic graph.
- Binary Boolean operations in the internal nodes ("gates").
- Sources (in-degree zero): inputs, or variables.
- Outputs: designated nodes.

Example: **Full Adder**. Three input bits.

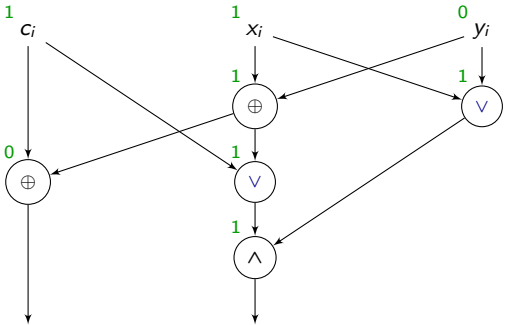


Background 2: Boolean circuits (non-uniform computations)

As viewed by Computer Scientists

- A directed acyclic graph.
- Binary Boolean operations in the internal nodes ("gates").
- Sources (in-degree zero): inputs, or variables.
- Outputs: designated nodes.
- Formalizes computation on fixed-length inputs (think about CPU).
- Time $\sim$ #nodes (for 1 CPU).

Example: **Full Adder**. Three input bits.

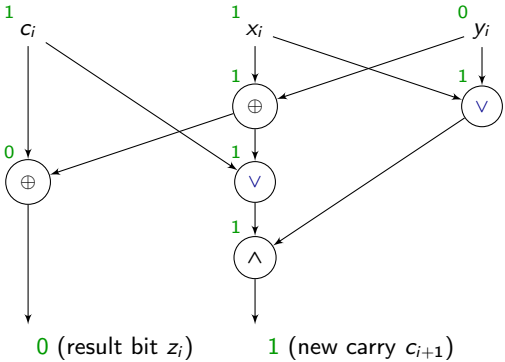


Background 2: Boolean circuits (non-uniform computations)

As viewed by Computer Scientists

- A directed acyclic graph.
- Binary Boolean operations in the internal nodes ("gates").
- Sources (in-degree zero): inputs, or variables.
- Outputs: designated nodes.
- Formalizes computation on fixed-length inputs (think about CPU).
- Time $\sim$ #nodes (for 1 CPU).

Example: **Full Adder**. Three input bits.



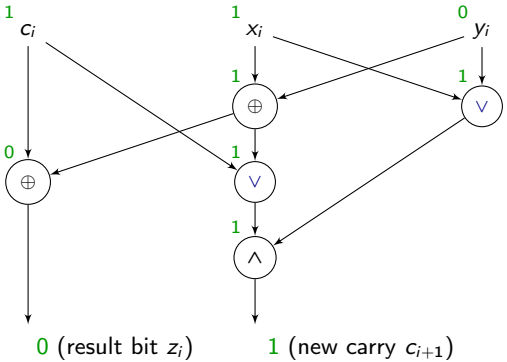
To work with inputs of arbitrary length, one needs an infinite family $\{C_0, C_1, C_2, C_3, \ldots\}$, where the circuit C_i has input length i. This is called non-uniform computation.

Background 2: Boolean circuits (non-uniform computations)

As viewed by Computer Scientists

- A directed acyclic graph.
- Binary Boolean operations in the internal nodes ("gates").
- Sources (in-degree zero): inputs, or variables.
- Outputs: designated nodes.
- Formalizes computation on fixed-length inputs (think about CPU).
- Time $\sim$ #nodes (for 1 CPU).

Example: **Full Adder**. Three input bits.



To work with inputs of arbitrary length, one needs an infinite family $\{C_0, C_1, C_2, C_3, \ldots\}$, where the circuit C_i has input length i. This is called non-uniform computation.

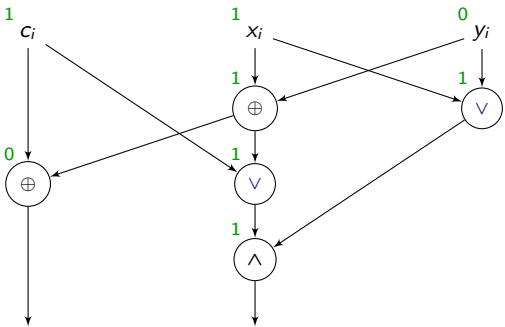
For each input size ("key length") they have a separate "algorithm".

Background 2: Boolean circuits (non-uniform computations)

As viewed by Computer Scientists

- A directed acyclic graph.
- Binary Boolean operations in the internal nodes ("gates").
- Sources (in-degree zero): inputs, or variables.
- Outputs: designated nodes.
- Formalizes computation on fixed-length inputs (think about CPU).
- Time $\sim$ #nodes (for 1 CPU).

Example: **Full Adder**. Three input bits.



To work with inputs of arbitrary length, one needs an infinite family $\{C_0, C_1, C_2, C_3, \ldots\}$, where the circuit C_i has input length i. This is called non-uniform computation.

For each input size ("key length") they have a separate "algorithm".

If given i, you can efficiently generate C_i, this is equivalent to computing in the usual (uniform) way.

Background 3: Probability Distribution

Probability Distribution:

Gives us right to write $\Pr\{\ldots\}$.
Gives us probability that a random variable
gets a specific value (discrete) or a value in a certain set (continuous).

Background 3: Probability Distribution

Probability Distribution:

Gives us right to write $\Pr\{\ldots\}$.
Gives us probability that a random variable
gets a specific value (discrete) or a value in a certain set (continuous).

Important Distributions:

Uniform (over $\{0,1\}^n$): $\Pr_{x \leftarrow U_n}\{x = x^\star\} = \frac{1}{2^n}$ (for every fixed $x^\star$).

Polynomial-time samplable: by polynomial-time algorithm A that outputs $A(r)$ given $r \leftarrow U_n$.

Background 3: Probability Distribution

Probability Distribution:

Gives us right to write $\Pr\{\ldots\}$.
Gives us probability that a random variable
gets a specific value (discrete) or a value in a certain set (continuous).

Important Distributions:

Uniform (over $\{0,1\}^n$): $\Pr_{x \leftarrow U_n}\{x = x^\star\} = \frac{1}{2^n}$ (for every fixed $x^\star$).

Polynomial-time samplable: by polynomial-time algorithm A that outputs $A(r)$ given $r \leftarrow U_n$.

Example

$A(r)$: `given a bit string` x`,`
`-- interpret it as an` n`-bit integer and compute its square` x^2`,`
`-- output the middle` n `bits of` x^2`.`

- Is the distribution generated by A uniform?
- Is it easy to distinguish it from uniform?
- Is it easy to compute r given $A(r)$?

Is Cryptography Possible?

Impagliazzo's Worlds

Russell Impagliazzo

(taken at CSR-2006, SPb, by A. Kulikov)

Algorithmica

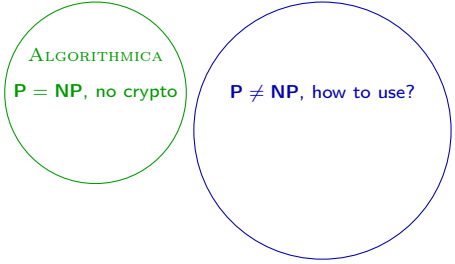
P = **NP**, no crypto

Is Cryptography Possible?

Impagliazzo's Worlds

Russell Impagliazzo

(taken at CSR-2006, SPb, by A. Kulikov)

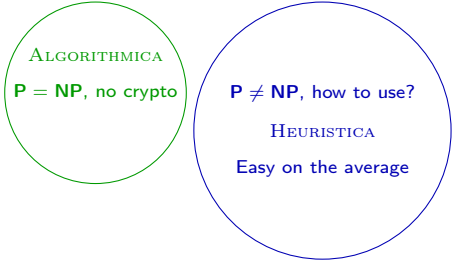


Is Cryptography Possible?

Impagliazzo's Worlds

Russell Impagliazzo

(taken at CSR-2006, SPb, by A. Kulikov)

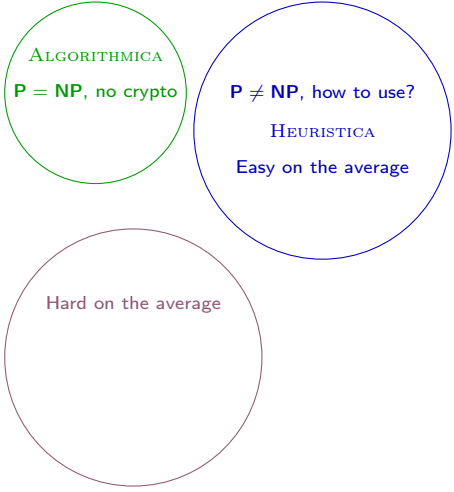


Is Cryptography Possible?

Impagliazzo's Worlds

Russell Impagliazzo

(taken at CSR-2006, SPb, by A. Kulikov)



Is Cryptography Possible?

Impagliazzo's Worlds

Russell Impagliazzo

(taken at CSR-2006, SPb, by A. Kulikov)

Algorithmica

P = NP, no crypto

P ≠ NP, how to use?

Heuristica

Easy on the average

Hard on the average

. . . also hard to generate

Pessiland

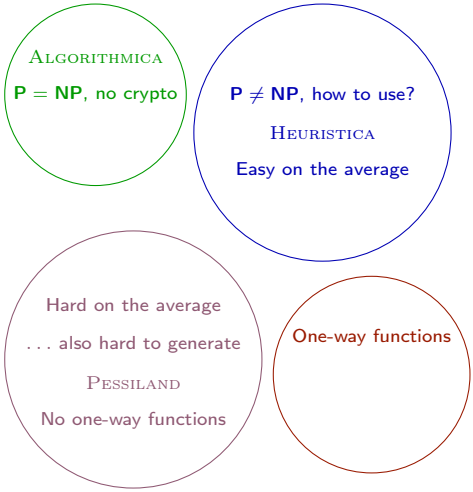
No one-way functions

Is Cryptography Possible?

Impagliazzo's Worlds

Russell Impagliazzo

(taken at CSR-2006, SPb, by A. Kulikov)



Is Cryptography Possible?

Impagliazzo's Worlds

Russell Impagliazzo

(taken at CSR-2006, SPb, by A. Kulikov)

Algorithmica

P = NP, no crypto

P ≠ NP, how to use?

Heuristica

Easy on the average

Hard on the average

. . . also hard to generate

Pessiland

No one-way functions

One-way functions

Minicrypt

digital signatures

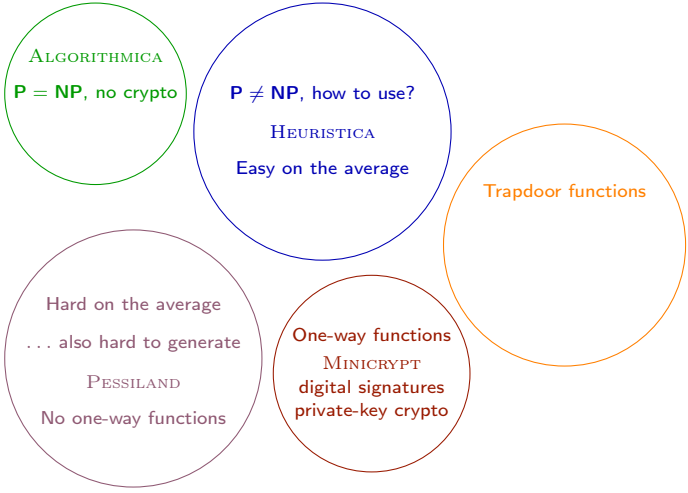
private-key crypto

Is Cryptography Possible?

Impagliazzo's Worlds

Russell Impagliazzo

(taken at CSR-2006, SPb, by A. Kulikov)



Is Cryptography Possible?

Impagliazzo's Worlds

Russell Impagliazzo

(taken at CSR-2006, SPb, by A. Kulikov)

Algorithmica

P = NP, no crypto

P ≠ NP, how to use?

Heuristica

Easy on the average

Trapdoor functions

Cryptomania

public-key encryption

digital money

Hard on the average

. . . also hard to generate

Pessiland

No one-way functions

One-way functions

Minicrypt

digital signatures

private-key crypto

Takeaway (Summary)

It was mostly an introductory lecture. Main points:

- Cryptographic tasks (overview).
- Non-uniform adversary: has a separate program for each "key length".

Coming next:

- One-way functions (and, eventually, rigorous definitions).

Questions?

- Drop me an email: edward.a.hirsch@gmail.com