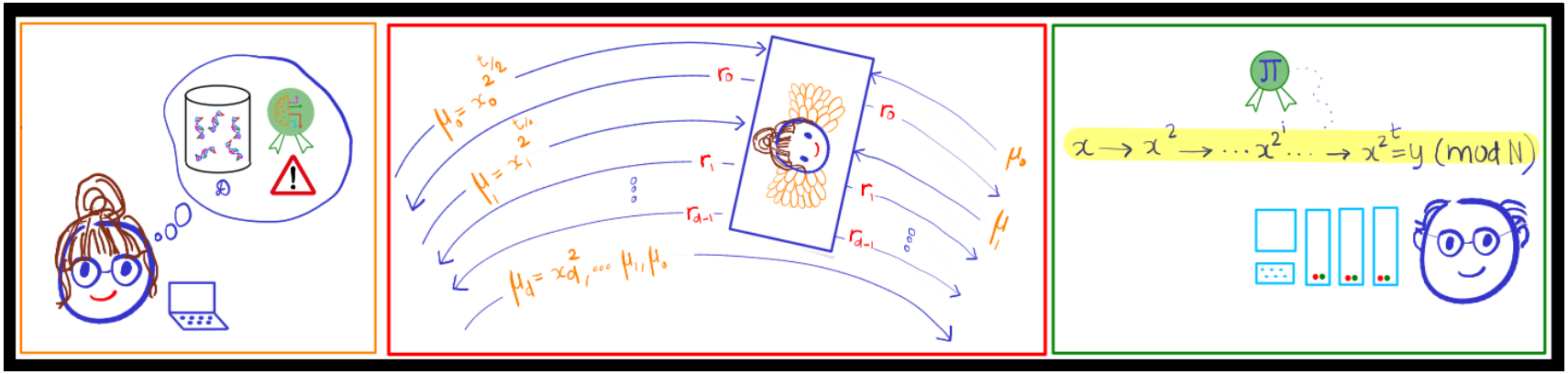


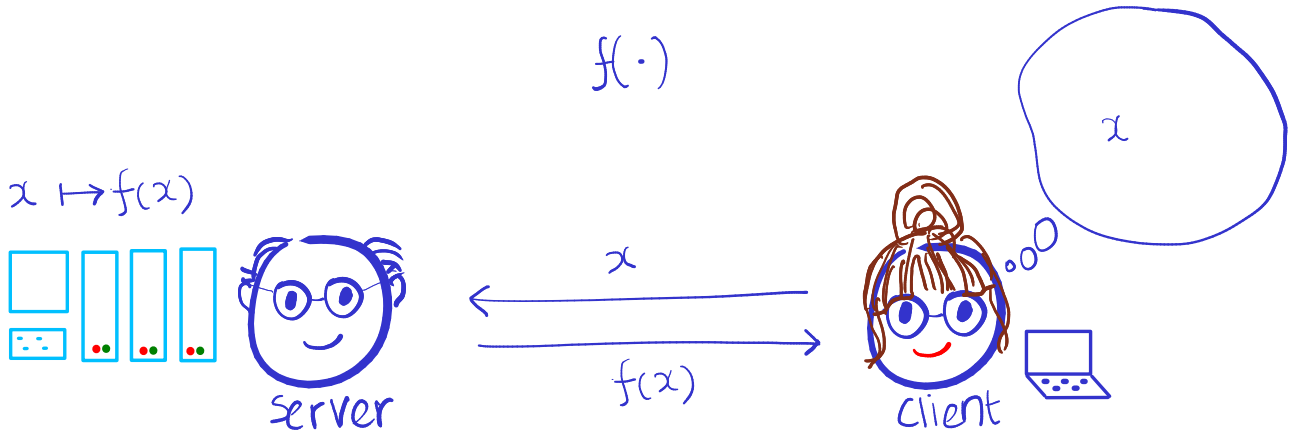
CS783: Theoretical Foundations of Cryptography

Lecture 20 (18/Oct/24)

Instructor: Chethan Kamath

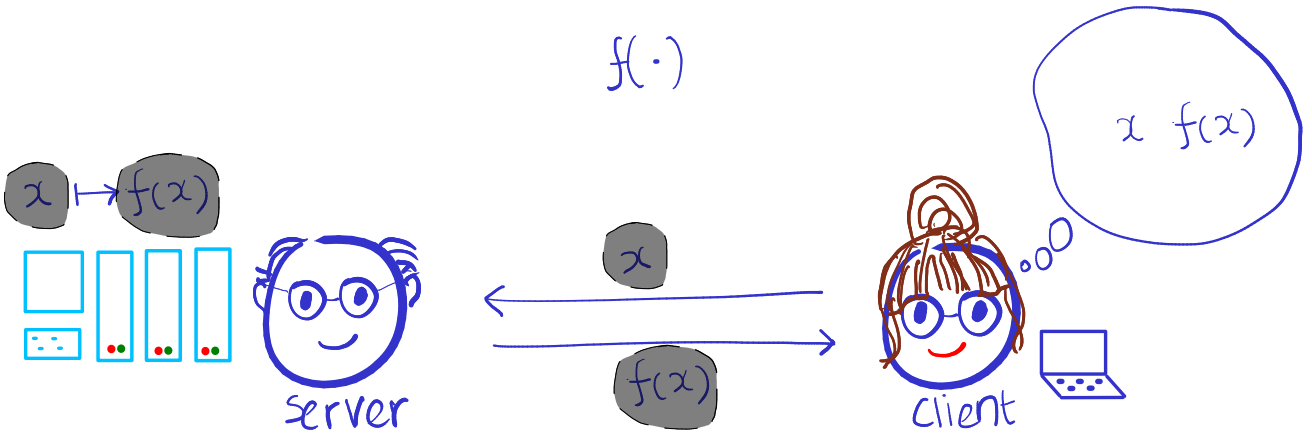
Recall from Last Lecture

- Task 7: secure outsourcing in the client-server setting
- Task 7.a: private outsourcing in the client-server setting



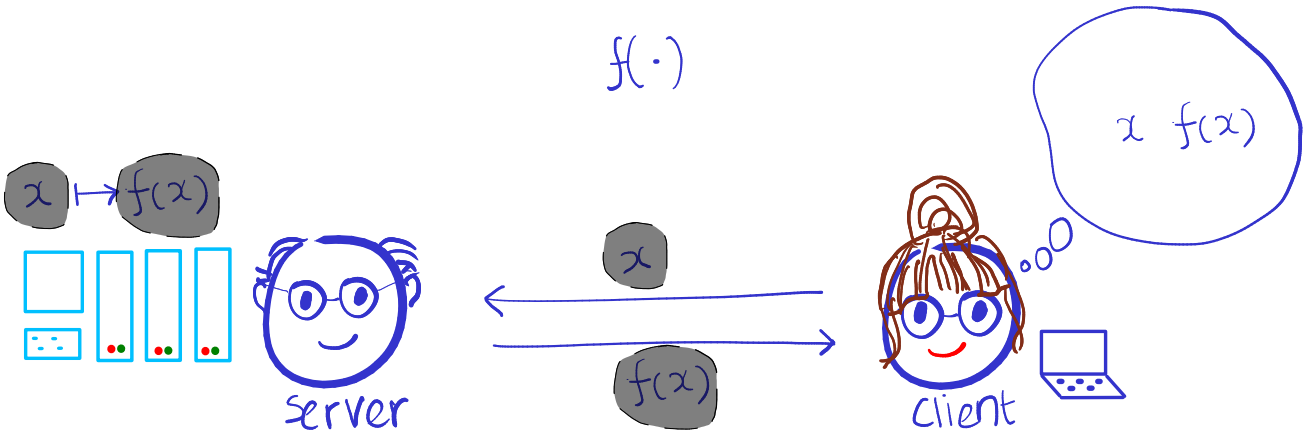
Recall from Last Lecture

- Task 7: secure outsourcing in the client-server setting
- Task 7.a: private outsourcing in the client-server setting



Recall from Last Lecture

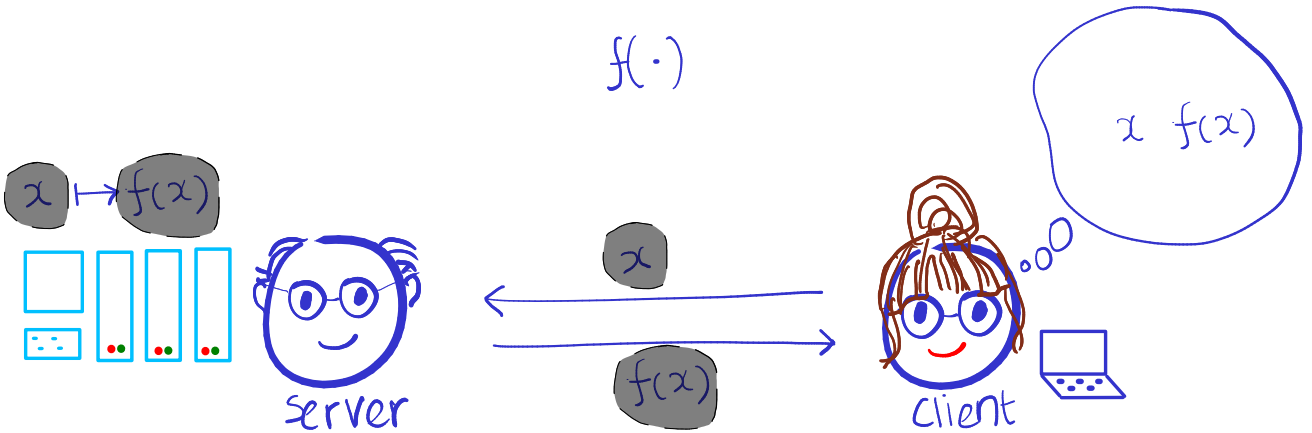
- Task 7: secure outsourcing in the client-server setting
- Task 7.a: private outsourcing in the client-server setting



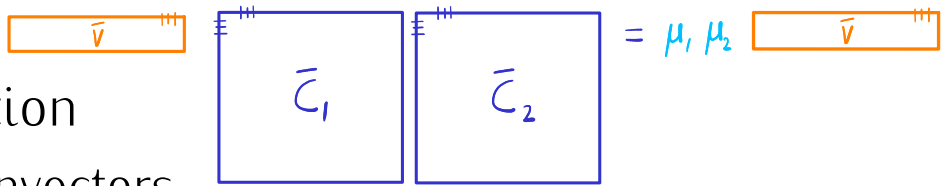
- Key tool: Fully homomorphic encryption (FHE)
  - FHE → private outsourcing of computation
  - In fact: FHE → 2PC of arbitrary functions

Recall from Last Lecture

- Task 7: secure outsourcing in the client-server setting
- Task 7.a: private outsourcing in the client-server setting



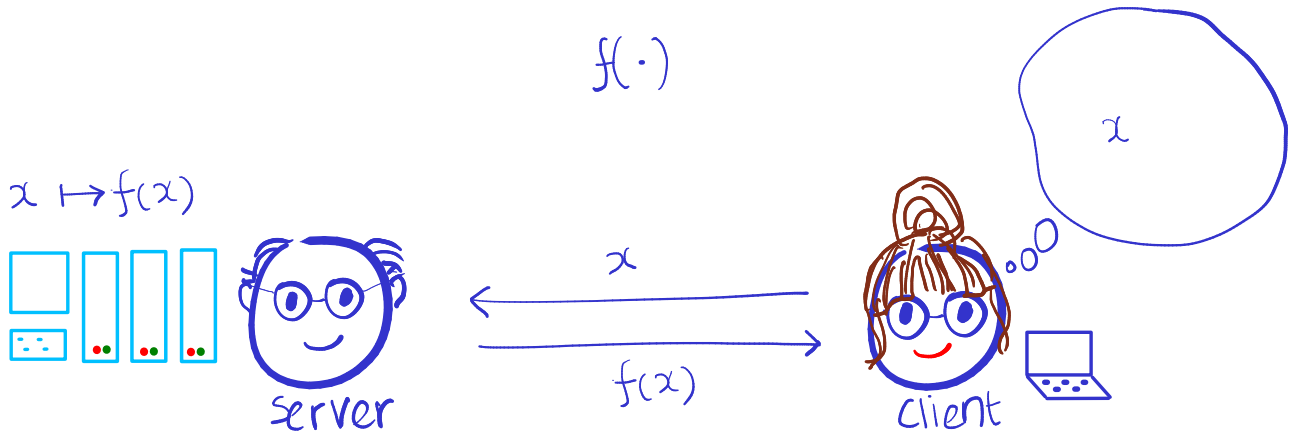
- Key tool: Fully homomorphic encryption (FHE)
  - FHE → private outsourcing of computation
  - In fact: FHE → 2PC of arbitrary functions



- GSW FHE from LWE assumption
  - Key idea: approximate eigenvectors
  - Similar idea used in approximate key exchange from LWE

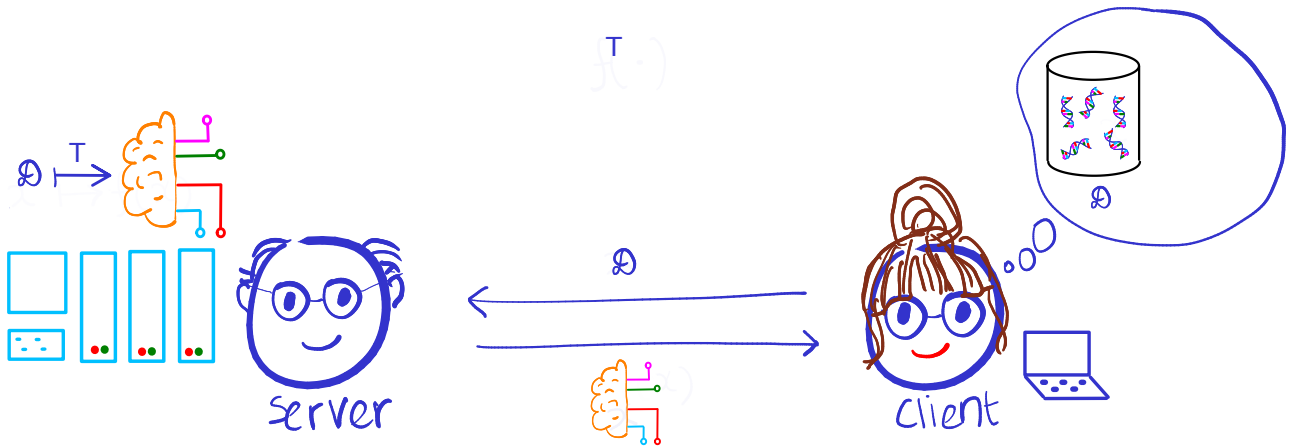
Plan for Today's Lecture...

- Task 7: secure outsourcing in the client-server setting



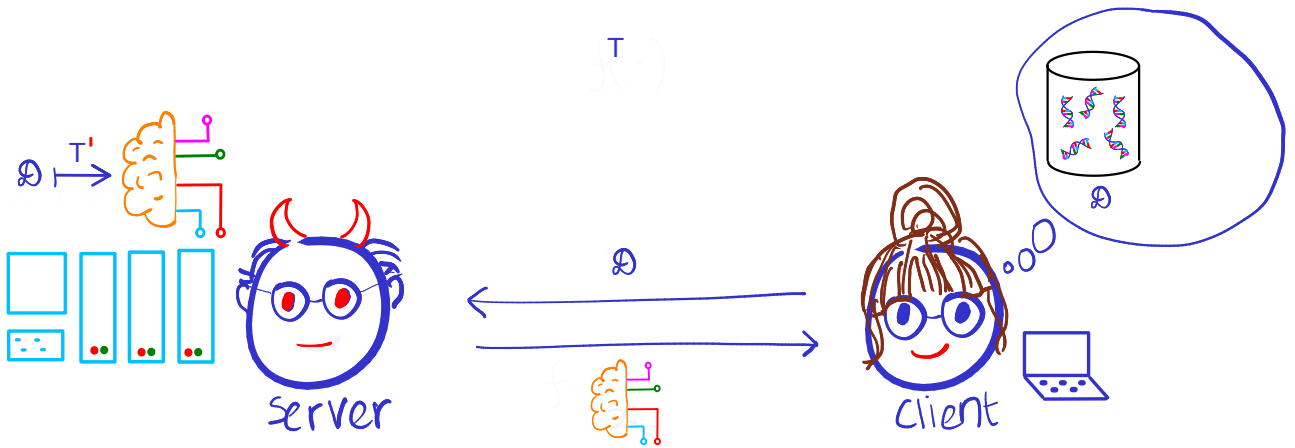
Plan for Today's Lecture...

- Task 7: secure outsourcing in the client-server setting



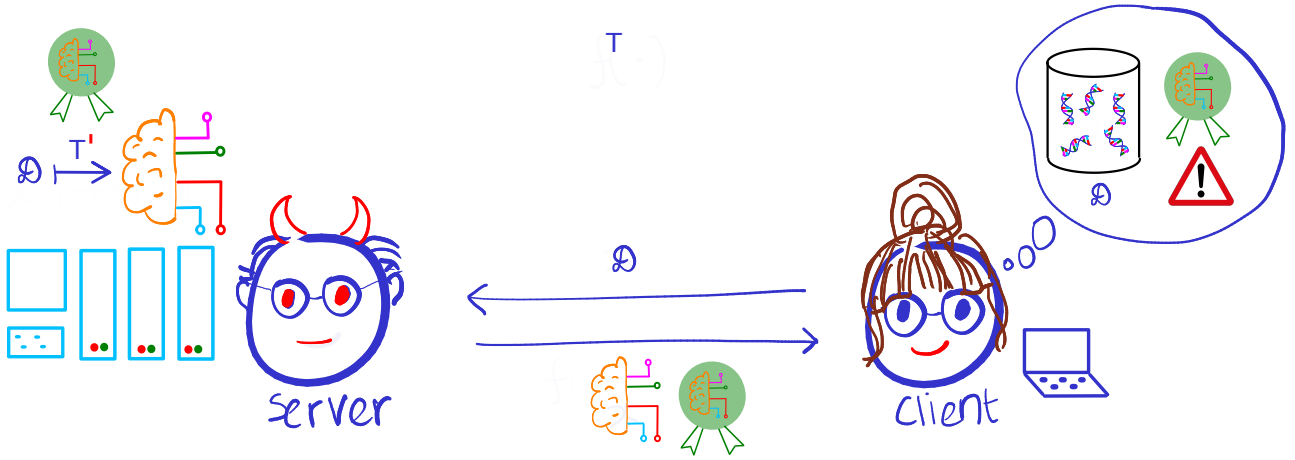
Plan for Today's Lecture...

- Task 7: secure outsourcing in the client-server setting
- Task 7.b: *verifiable* outsourcing in the client-server setting



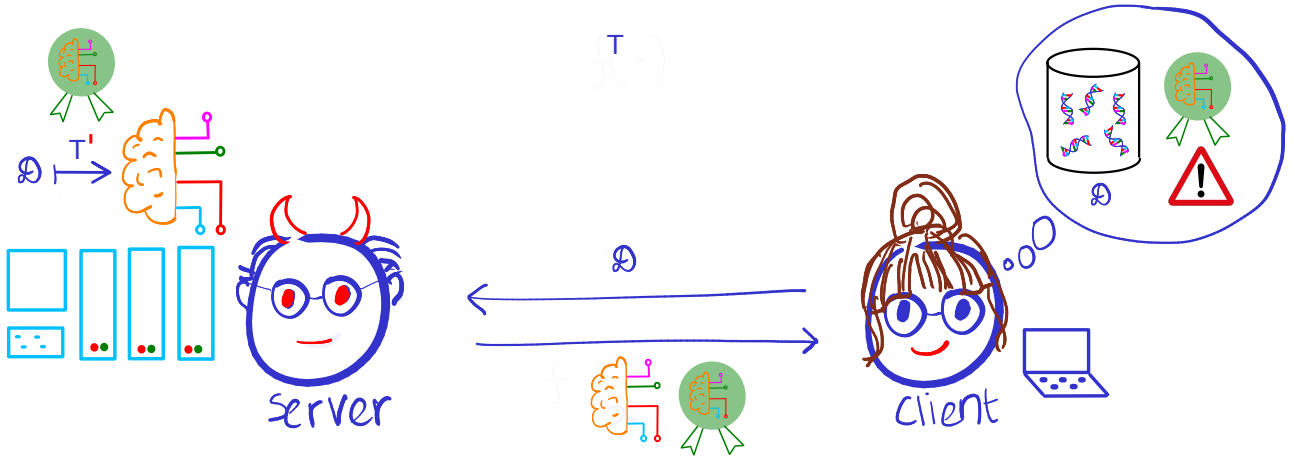
Plan for Today's Lecture...

- Task 7: secure outsourcing in the client-server setting
- Task 7.b: *verifiable* outsourcing in the client-server setting
  - Verifying faster than recomputing

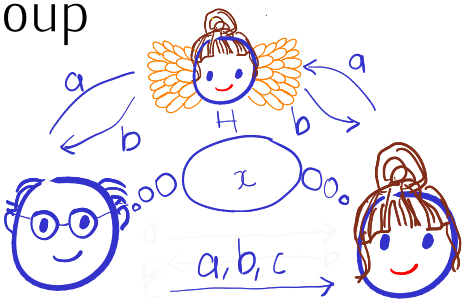


Plan for Today's Lecture...

- Task 7: secure outsourcing in the client-server setting
- Task 7.b: *verifiable* outsourcing in the client-server setting
  - Verifying faster than recomputing



- Key tool: succinct non-interactive argument (SNARG)
- SNARG for repeated squaring function in RSA group
  - Pietrzak's interactive protocol
  - Remove interaction using Fiat-Shamir transform



Plan for Today's Lecture...

General *template*:

1. Identify the task → Task 7.b: verifiable outsourcing
2. Come up with precise threat model M (a.k.a security model)
   - Adversary/Attack: What are the adversary's capabilities?
   - Security Goal: What does it mean to be secure?
3. Construct a scheme Π
4. Formally prove that Π in secure in model M

Plan for Today's Lecture...

General *template*:

1. Identify the task
2. Come up with precise threat model M (a.k.a security model)
   - Adversary/Attack: What are the adversary's capabilities?
   - Security Goal: What does it mean to be secure?
3. Construct a scheme Π
4. Formally prove that Π in secure in model M

Task 7.b: verifiable outsourcing

malicious server

computational soundness

Plan for Today's Lecture...

General *template*:

1. Identify the task → Task 7.b: verifiable outsourcing
2. Come up with precise threat model M (a.k.a security model)
   - Adversary/Attack: What are the adversary's capabilities? → malicious server
   - Security Goal: What does it mean to be secure? → computational soundness
3. Construct a scheme Π → SNARG-based
4. Formally prove that Π in secure in model M
   → in random-oracle model

Plan for Today's Lecture

1. Verifiable Outsourcing of Computation
   - Succinct Non-Interactive Argument (SNARG)

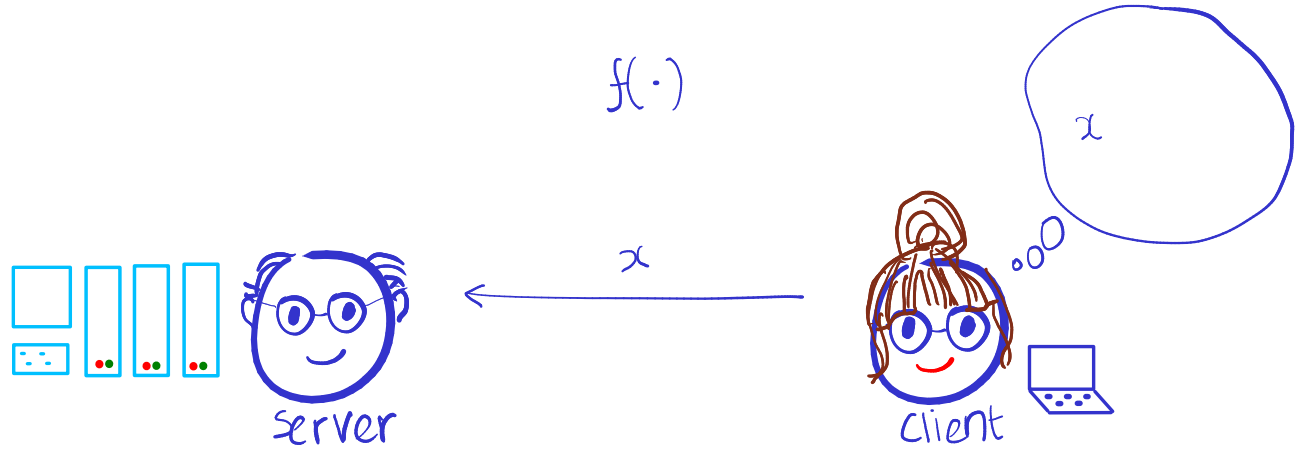
2. SNARG for Repeated Squaring
   - Step I: Interactive Proof for Repeated Squaring
   - Step II: Remove Interaction via Fiat-Shamir Transform

Plan for Today's Lecture

1. Verifiable Outsourcing of Computation
   - Succinct Non-Interactive Argument (SNARG)

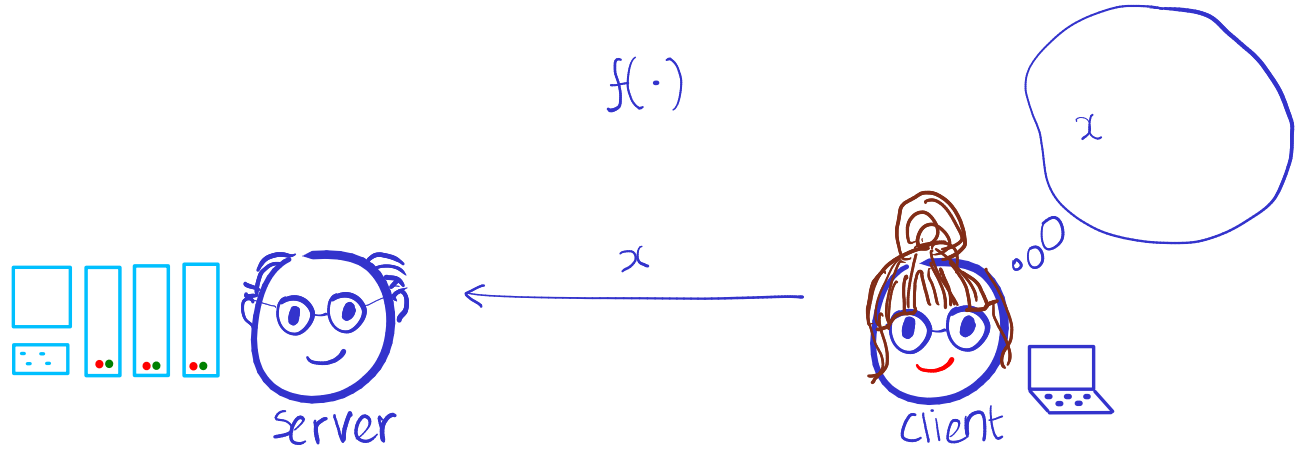
2. SNARG for Repeated Squaring
   - Step I: Interactive Proof for Repeated Squaring
   - Step II: Remove Interaction via Fiat-Shamir Transform

Verifiable Outsourcing: Setting and Requirements



- Setting:
  - Client is resource-constrained and server is powerful
  - Function f : $\mathcal{X} \to \mathcal{Y}$ known to both client and server
  - Client wants to get f evaluated on an input $x \in \mathcal{X}$

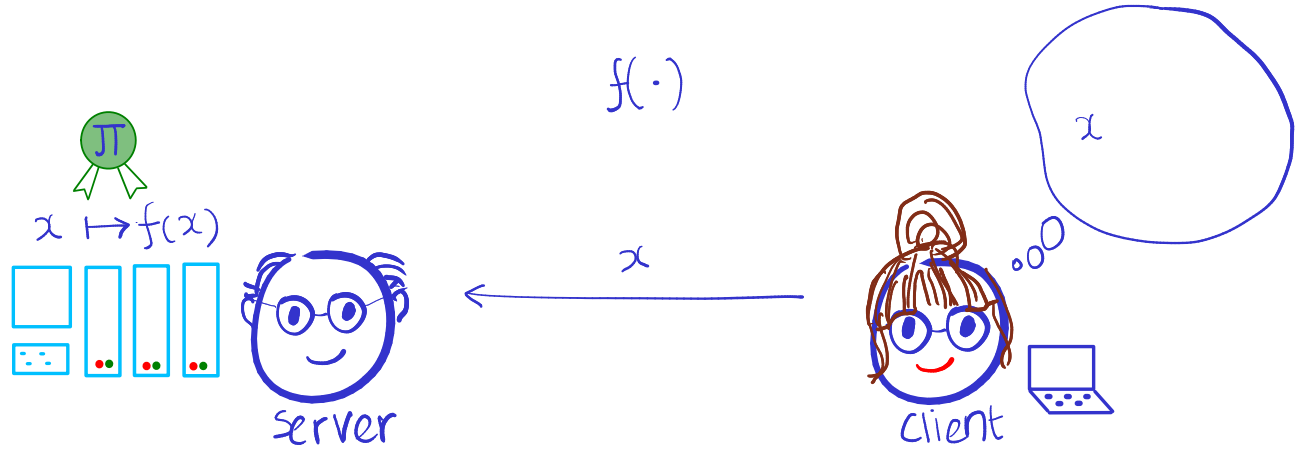
Verifiable Outsourcing: Setting and Requirements



- Setting:
  - Client is resource-constrained and server is powerful
  - Function f : $\mathcal{X} \to \mathcal{Y}$ known to both client and server
  - Client wants to get f evaluated on an input $x \in \mathcal{X}$

DTIME(t) → computation

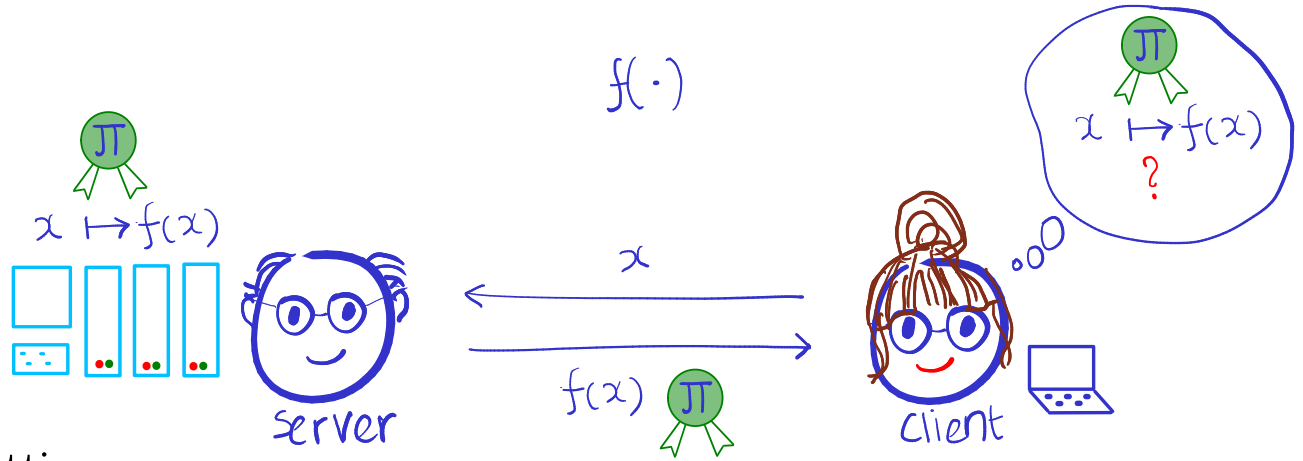
Verifiable Outsourcing: Setting and Requirements



- Setting:
  - Client is resource-constrained and server is powerful
  - Function $f : \mathcal{X} \to \mathcal{Y}$ known to both client and server
  - Client wants to get f evaluated on an input $x \in \mathcal{X}$
  - Server computes output $y = f(x)$ and a proof π

DTIME(t) → computation

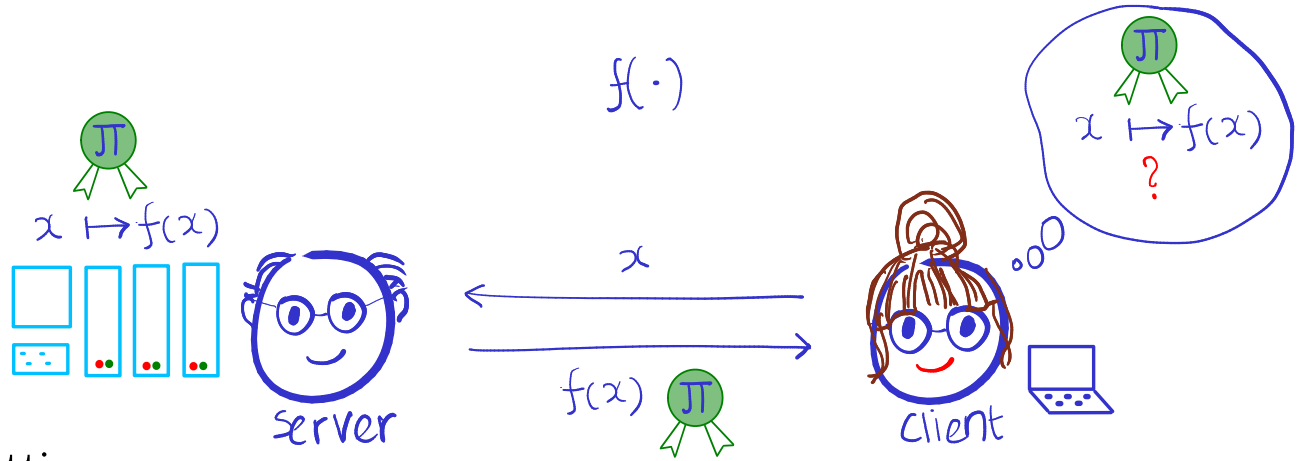
Verifiable Outsourcing: Setting and Requirements



- Setting:
  - Client is resource-constrained and server is powerful
  - Function $f : \mathcal{X} \to \mathcal{Y}$ known to both client and server
  - Client wants to get f evaluated on an input $x \in \mathcal{X}$
  - Server computes output $y = f(x)$ and a proof π
  - Server sends (y, π) to client, who uses π to verify that $y = f(x)$

DTIME(t) computation

Verifiable Outsourcing: Setting and Requirements

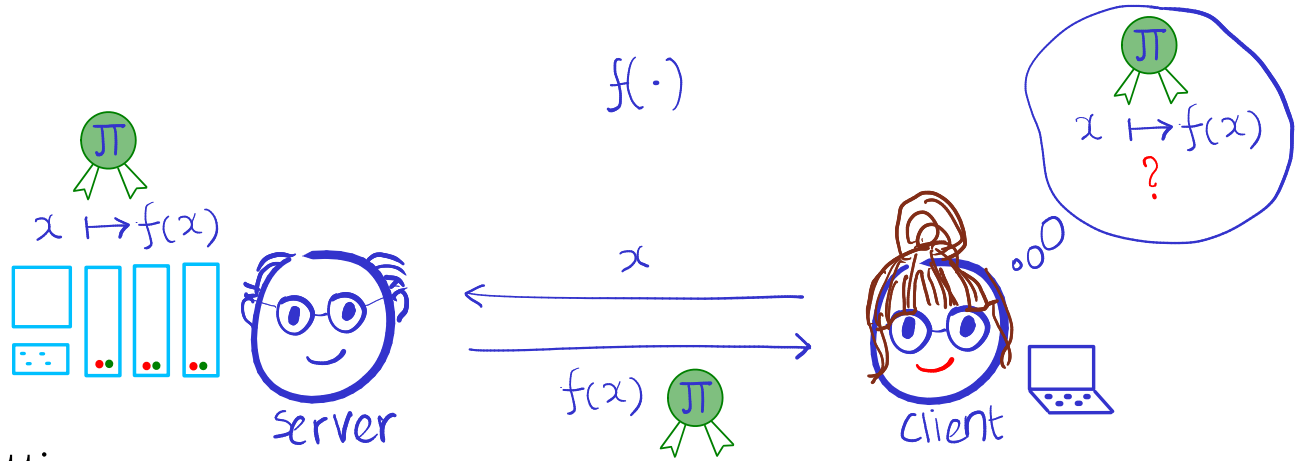


- Setting:
  - Client is resource-constrained and server is powerful
  - Function $f : \mathcal{X} \to \mathcal{Y}$ known to both client and server
  - Client wants to get f evaluated on an input $x \in \mathcal{X}$
  - Server computes output $y = f(x)$ and a proof π
  - Server sends (y, π) to client, who uses π to verify that $y = f(x)$

DTIME(t) → computation

? Requirements:

Verifiable Outsourcing: Setting and Requirements

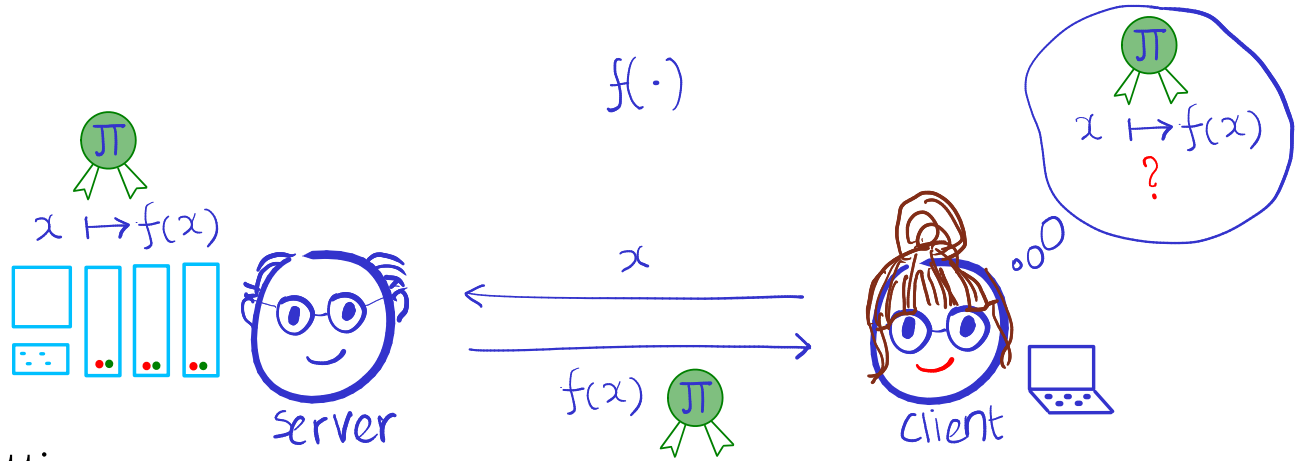


- Setting:
  - Client is resource-constrained and server is powerful
  - Function f : $\mathcal{X} \to \mathcal{Y}$ known to both client and server
  - Client wants to get f evaluated on an input $x \in \mathcal{X}$
  - Server computes output $y = f(x)$ and a proof π
  - Server sends (y, π) to client, who uses π to verify that $y = f(x)$

DTIME(t) computation

- Requirements:
  - Complete: if $y = f(x)$ then client should *accept* π
  - Sound: if $y \neq f(x)$ then client should *reject* π

Verifiable Outsourcing: Setting and Requirements



- Setting:
  - Client is resource-constrained and server is powerful
  - Function $f : \mathcal{X} \to \mathcal{Y}$ known to both client and server
  - Client wants to get f evaluated on an input $x \in \mathcal{X}$
  - Server computes output $y = f(x)$ and a proof π
  - Server sends (y, π) to client, who uses π to verify that $y = f(x)$

DTIME(t) computation

- Requirements:
  - Complete: if $y = f(x)$ then client should *accept* π
  - Sound: if $y \neq f(x)$ then client should *reject* π
  - Efficient:
    - Server-side: overhead to compute π should be low
    - Client-side: verify π much faster than recomputing $f(x)$

ideally $O(t)$

ideally $O(\log(t))$

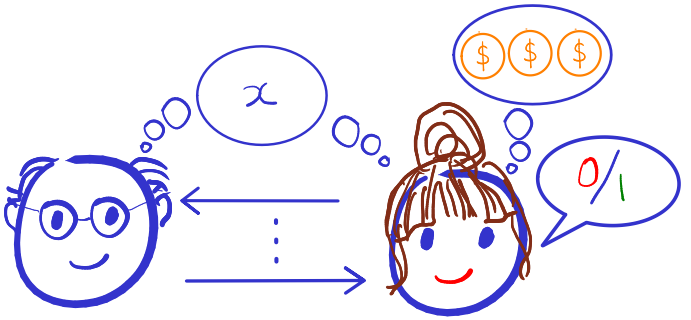
Plan for this Session

1. Verifiable Outsourcing of Computation
   - Succinct Non-Interactive Argument (SNARG)

2. SNARG for Repeated Squaring
   - Step I: Interactive Proof for Repeated Squaring
   - Step II: Remove Interaction via Fiat-Shamir Transform

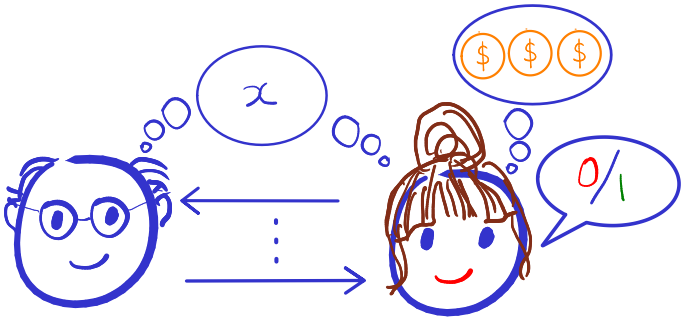
Recall Interactive Proofs

- Difference from NP proofs:
  - 1 Verifier V is *randomised*
  - 2 Prover P and V *interact* and V accepts/rejects in the end



Recall Interactive Proofs

- Difference from NP proofs:
  - 1 Verifier V is *randomised*
  - 2 Prover P and V *interact* and V accepts/rejects in the end



Defintion 1 (Interactive proof for a language $\mathcal{L}$)

An interactive protocol (P, V) *for* $\mathcal{L}$ *that is:*

completeness error $\varepsilon_c(n)$

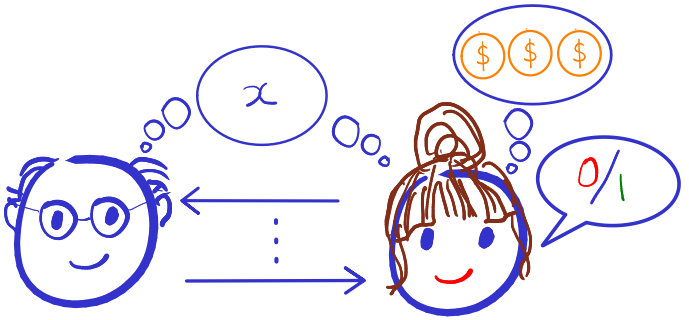
- *Complete: for every* $x \in \mathcal{L}$, $\Pr[1 \leftarrow \langle \mathsf{P}, \mathsf{V} \rangle(x)] \geq 1 - 1/3$
- *Sound: for every* $x \notin \mathcal{L}$ *and malicious prover* P^*,

soundness error $\varepsilon_s(n)$

$$\Pr[1 \leftarrow \langle \mathsf{P}^*, \mathsf{V} \rangle(x)] \leq 1/3$$

Recall Interactive Proofs

- Difference from NP proofs:
  1. Verifier V is *randomised*
  2. Prover P and V *interact* and V accepts/rejects in the end



Defintion 1 (Interactive proof for a language $\mathcal{L}$)

An interactive protocol (P, V) *for* $\mathcal{L}$ *that is:*

completeness error $\varepsilon_c(n)$

- *Complete: for every* $x \in \mathcal{L}$, $\Pr[1 \leftarrow \langle \mathsf{P}, \mathsf{V} \rangle(x)] \geq 1 - 1/3$
- *Sound: for every* $x \notin \mathcal{L}$ *and malicious prover* P^*,

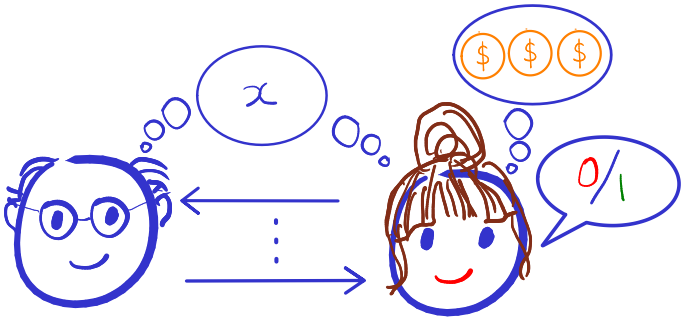
soundness error $\varepsilon_s(n)$

$$\Pr[1 \leftarrow \langle \mathsf{P}^*, \mathsf{V} \rangle(x)] \leq 1/3$$

- We saw (zero-knowledge) IP for GI and GNI

Recall Interactive Proofs

- Difference from NP proofs:
  - 1 Verifier V is *randomised*
  - 2 Prover P and V *interact* and V accepts/rejects in the end



Defintion 1 (Interactive proof for a language $\mathcal{L}$)

An interactive protocol (P, V) *for* $\mathcal{L}$ *that is:*

completeness error $\varepsilon_c(n)$

- *Complete: for every* $x \in \mathcal{L}$, $\Pr[1 \leftarrow \langle \mathsf{P}, \mathsf{V} \rangle(x)] \geq 1 - 1/3$
- *Sound: for every* $x \notin \mathcal{L}$ *and malicious prover* P^*,

soundness error $\varepsilon_s(n)$

$$\Pr[1 \leftarrow \langle \mathsf{P}^*, \mathsf{V} \rangle(x)] \leq 1/3$$

- We saw (zero-knowledge) IP for GI and GNI
  - ? What was the communication complexity for those IPs?

(size of transcript)

Let's Define Succinct, Non-Interactive Proofs for $\mathcal{L}_f$...

- Recall our requirements for verifiable outsourcing:
  - → Non-interactive: server sends π in one go
  - Complete: if $y = f(x)$ then client should *accept* π
  - Sound: if $y \neq f(x)$ then client should *reject* π
  - Succinct: verifying π should be faster than recomputing $f(x)$

Let's Define Succinct, Non-Interactive Proofs for $\mathcal{L}_f$...

$\{(x,y) : y = f(x)\}$

- Recall our requirements for verifiable outsourcing:
  - Non-interactive: server sends π in one go
  - Complete: if $y = f(x)$ then client should *accept* π
  - Sound: if $y \neq f(x)$ then client should *reject* π
  - Succinct: verifying π should be faster than recomputing $f(x)$

Defintion 2 (Succinct Non-Interactive Proof, first attempt)

A non-interactive protocol (P, V) *for* $\mathcal{L}_f$ *that is:*

Let's Define Succinct, Non-Interactive Proofs for $\mathcal{L}_f$...

$\{(x,y) : y = f(x)\}$

- Recall our requirements for verifiable outsourcing:
  - → Non-interactive: server sends π in one go
  - Complete: if $y = f(x)$ then client should *accept* π
  - Sound: if $y \neq f(x)$ then client should *reject* π
  - Succinct: verifying π should be faster than recomputing $f(x)$

Defintion 2 (Succinct Non-Interactive Proof, first attempt)

A non-interactive protocol (P, V) *for* $\mathcal{L}_f$ *that is:*

- *Complete: for every* $(x, y) \in \mathcal{L}_f$: $\Pr_{\pi \leftarrow P(x,y)}[V(x,y,\pi) = 1] = 1$
- *Sound: for every* $(x, y) \notin \mathcal{L}_f$ *and malicious prover* P^*, $\Pr_{\pi \leftarrow P^*(x,y)}[V(x,y,\pi) = 1] = 0$
- *Succinct:* $|\pi| \leq o(t)$ *and verifier runs in time* $o(t)$

Let's Define Succinct, Non-Interactive Proofs for $\mathcal{L}_f$...

$\{(x,y) : y = f(x)\}$

- Recall our requirements for verifiable outsourcing:
  - Non-interactive: server sends π in one go
  - Complete: if $y = f(x)$ then client should *accept* π
  - Sound: if $y \neq f(x)$ then client should *reject* π
  - Succinct: verifying π should be faster than recomputing $f(x)$

Defintion 2 (Succinct Non-Interactive Proof, first attempt)

A non-interactive protocol (P, V) *for* $\mathcal{L}_f$ *that is:*

- *Complete: for every* $(x, y) \in \mathcal{L}_f$: $\Pr_{\pi \leftarrow P(x,y)}[V(x,y,\pi) = 1] = 1$
- *Sound: for every* $(x, y) \notin \mathcal{L}_f$ *and malicious prover* P^*, $\Pr_{\pi \leftarrow P^*(x,y)}[V(x,y,\pi) = 1] = 0$
- *Succinct:* $|\pi| \leq o(t)$ *and verifier runs in time* $o(t)$ ← ideally $O(\log(t))$

Let's Define Succinct, Non-Interactive Proofs for $\mathcal{L}_f$...

$\{(x,y) : y = f(x)\}$

- Recall our requirements for verifiable outsourcing:
  - Non-interactive: server sends π in one go
  - Complete: if $y = f(x)$ then client should *accept* π
  - Sound: if $y \neq f(x)$ then client should *reject* π
  - Succinct: verifying π should be faster than recomputing $f(x)$

Defintion 2 (Succinct Non-Interactive Proof, first attempt)

A non-interactive protocol (P, V) *for* $\mathcal{L}_f$ *that is:*

- *Complete: for every* $(x, y) \in \mathcal{L}_f$: $\Pr_{\pi \leftarrow P(x,y)}[V(x,y,\pi) = 1] = 1$
- *Sound: for every* $(x, y) \notin \mathcal{L}_f$ *and malicious prover* P^*, $\Pr_{\pi \leftarrow P^*(x,y)}[V(x,y,\pi) = 1] = 0$
- *Succinct:* $|\pi| \leq o(t)$ *and verifier runs in time* $o(t)$ ← ideally $O(\log(t))$

Exercise 1 (Problem with Defintion 2?

Let's Define Succinct, Non-Interactive Proofs for $\mathcal{L}_f$...

$\{(x,y) : y = f(x)\}$

- Recall our requirements for verifiable outsourcing:
  - → Non-interactive: server sends π in one go
  - Complete: if $y = f(x)$ then client should *accept* π
  - Sound: if $y \neq f(x)$ then client should *reject* π
  - Succinct: verifying π should be faster than recomputing $f(x)$

Defintion 2 (Succinct Non-Interactive Proof, first attempt)

A non-interactive protocol (P, V) *for* $\mathcal{L}_f$ *that is:*

- *Complete: for every* $(x, y) \in \mathcal{L}_f$: $\Pr_{\pi \leftarrow P(x,y)}[V(x,y,\pi) = 1] =$
- *Sound: for every* $(x, y) \notin \mathcal{L}_f$ *and malicious prover* P^*, $\Pr_{\pi \leftarrow P^*(x,y)}[V(x,y,\pi) = 1] = 0$
- *Succinct:* $|\pi| \leq o(t)$ *and verifier runs in time* $o(t)$ ← ideally $O(\log(t))$

Exercise 1 (Problem with Defintion 2? It is too strong)

Show that if $\mathcal{L}_f \in \mathsf{DTIME}(t)$ *has succinct non-interactive proof as per Defintion 2 then* $\mathsf{DTIME}(t) \subseteq \mathsf{NTIME}(o(t))$

Let's Define Succinct, Non-Interactive Proofs for $\mathcal{L}_f$...

- Recall our requirements for verifiable outsourcing:
  - → Non-interactive: server sends π in one go
  - Complete: if $y = f(x)$ then client should *accept* π
  - Sound: if $y \neq f(x)$ then client should *reject* π
  - Succinct: verifying π should be faster than recomputing $f(x)$

$\{(x,y) : y = f(x)\}$

Defintion 2 (Succinct Non-Interactive Proof, first attempt)

A non-interactive protocol (P, V) *for* $\mathcal{L}_f$ *that is:*

- *Complete: for every* $(x, y) \in \mathcal{L}_f$: $\Pr_{\pi \leftarrow P(x,y)}[V(x,y,\pi) = 1] \geq 1 - 1/3$
- *Sound: for every* $(x, y) \notin \mathcal{L}_f$ *and malicious prover* P^*, $\Pr_{\pi \leftarrow P^*(x,y)}[V(x,y,\pi) = 1] \leq 1/3$ Too strong even if
- *Succinct:* $|\pi| \leq o(t)$ *and verifier runs in time* $o(t)$ ← Ideally $O(\log(t))$

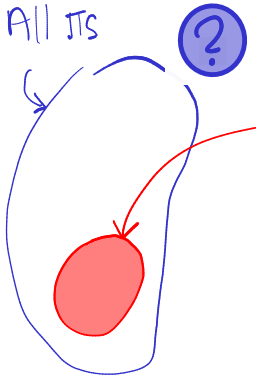
Exercise 1 (Problem with Defintion 2? It is too strong)

Show that if $\mathcal{L}_f \in \mathsf{DTIME}(t)$ *has succinct non-interactive proof as per Defintion 2 then* $\mathsf{DTIME}(t) \subseteq \mathsf{NTIME}(o(t))$

Let's Define Succinct, Non-Interactive Proofs for $\mathcal{L}_f$...

- The way around?

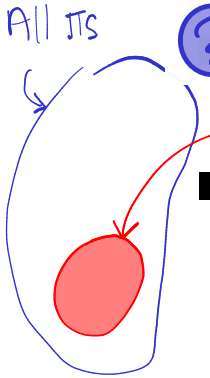
Let's Define Succinct, Non-Interactive Proofs for $\mathcal{L}_f$...



The way around? Relax to computational soundness=argument

- Bad proofs may exist but are computationally *hard* to find

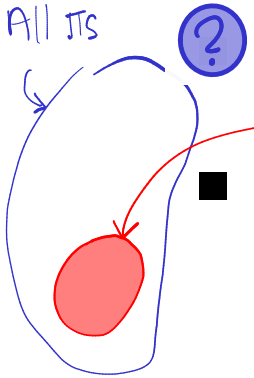
Let's Define Succinct, Non-Interactive Proofs for $\mathcal{L}_f$...



The way around? Relax to computational soundness=argument

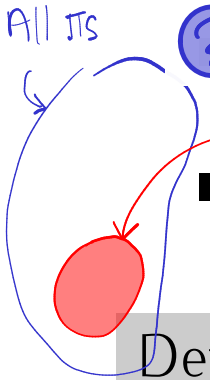
- Bad proofs may exist but are computationally *hard* to find
- Need source of comp. hardness to limit malicious prover

Let's Define Succinct, Non-Interactive Proofs for $\mathcal{L}_f$...



- The way around? Relax to computational soundness=argument
  - Bad proofs may exist but are computationally *hard* to find
- Need source of comp. hardness to limit malicious prover
  - Common random string model or random-oracle model (ROM)

Let's Define Succinct, Non-Interactive Proofs for $\mathcal{L}_f$...



- The way around? Relax to computational soundness=argument
  - Bad proofs may exist but are computationally *hard* to find
- Need source of comp. hardness to limit malicious prover
  - Common random string model or random-oracle model (ROM)

Defintion 3 (Succinct Non-Interactive Argument in ROM)

A non-interactive protocol $(\mathsf{P}^H, \mathsf{V}^H)$ *for* $\mathcal{L}_f$ *that is:*

- *Complete: for every* $(x, y) \in \mathcal{L}_f$ *and* $n \in \mathbb{N}$,

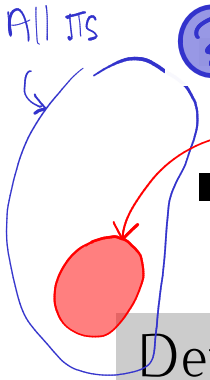
$$\Pr_{H,\pi \leftarrow P^H(1^n,x,y)}\left[V^H(x,y,\pi)=1\right]=1$$

- *Computationally sound: for every* $(x, y) \notin \mathcal{L}_f$ *and PPT malicious prover* P^**, the following is negligible*

$$\Pr_{H,\pi \leftarrow P^{*H}(1^n,x,y)}\left[V^H(x,y,\pi)=1\right]$$

- *Succinct:* $|\pi| \leq o(t) \cdot \mathsf{poly}(n)$ *verifier runs in time* $o(t) \cdot \mathsf{poly}(n)$

Let's Define Succinct, Non-Interactive Proofs for $\mathcal{L}_f$...



- The way around? Relax to computational soundness=argument
  - Bad proofs may exist but are computationally *hard* to find
- Need source of comp. hardness to limit malicious prover
  - Common random string model or random-oracle model (ROM)

Defintion 3 (Succinct Non-Interactive Argument in ROM)

A non-interactive protocol $(\mathsf{P}^H, \mathsf{V}^H)$ *for* $\mathcal{L}_f$ *that is:*

- *Complete: for every* $(x, y) \in \mathcal{L}_f$ *and* $n \in \mathbb{N}$,

$$\Pr_{H,\pi \leftarrow P^H(1^n,x,y)}\left[V^H(x,y,\pi)=1\right]=1$$

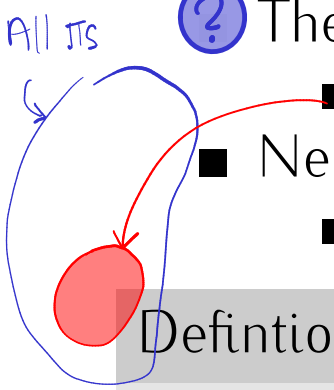
- *Computationally sound: for every* $(x, y) \notin \mathcal{L}_f$ *and PPT malicious prover* P^**, the following is negligible*

$$\Pr_{H,\pi \leftarrow P^{*H}(1^n,x,y)}\left[V^H(x,y,\pi)=1\right]$$

- *Succinct:* $|\pi| \leq o(t) \cdot \mathsf{poly}(n)$ *verifier runs in time* $o(t) \cdot \mathsf{poly}(n)$

$O(\log(t))$

Let's Define Succinct, Non-Interactive Proofs for $\mathcal{L}_f$...



- The way around? Relax to computational soundness=argument
  - Bad proofs may exist but are computationally *hard* to find
- Need source of comp. hardness to limit malicious prover
  - Common random string model or random-oracle model (ROM)

Defintion 3 (Succinct Non-Interactive Argument in ROM)

A non-interactive protocol $(\mathsf{P}^H, \mathsf{V}^H)$ *for* $\mathcal{L}_f$ *that is:*

- *Complete: for every* $(x, y) \in \mathcal{L}_f$ *and* $n \in \mathbb{N}$,

$$\Pr_{H,\pi \leftarrow P^H(1^n,x,y)}\left[V^H(x,y,\pi)=1\right]=1$$

- *Computationally sound: for every* $(x, y) \notin \mathcal{L}_f$ *and PPT malicious prover* P^**, the following is negligible*

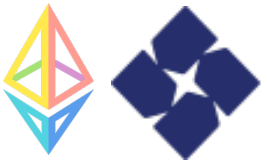
$$\Pr_{H,\pi \leftarrow P^{*H}(1^n,x,y)}\left[V^H(x,y,\pi)=1\right]$$

- *Succinct:* $|\pi| \leq o(t) \cdot \mathsf{poly}(n)$ *verifier runs in time* $o(t) \cdot \mathsf{poly}(n)$

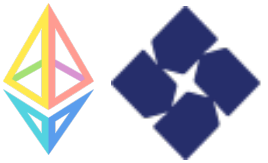
How to carry out verifiable outsourcing using SNARGs?

Where are SNARGs Useful?

- Compute as a service (same as *private* outsourcing)
- Blockchain: Ethereum, Starkware etc have deployed SNARGs

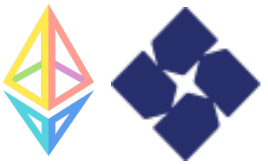


Where are SNARGs Useful?

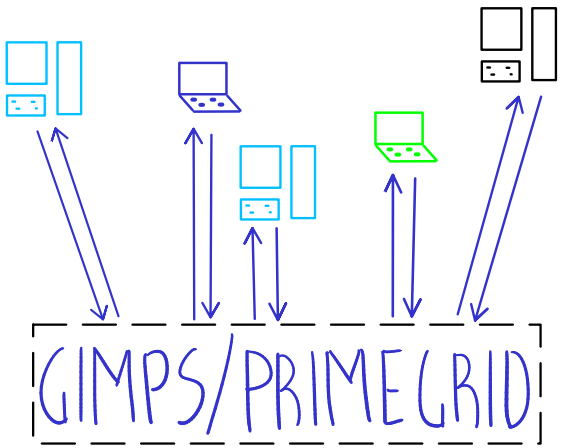


- Compute as a service (same as *private* outsourcing)
- Blockchain: Ethereum, Starkware etc have deployed SNARGs
- Great Internet Mersenne Prime Search (GIMPS)

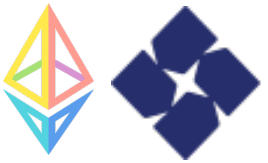
Where are SNARGs Useful?



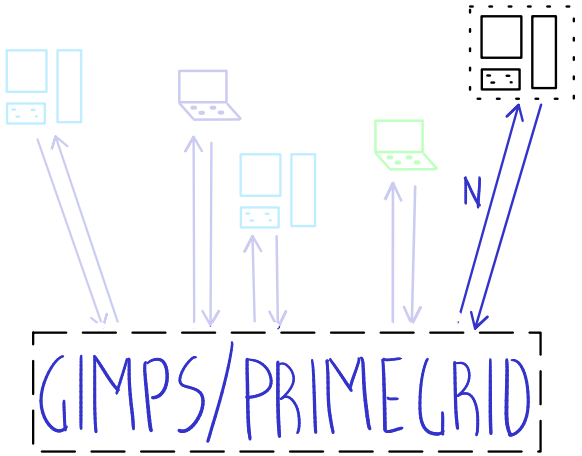
- Compute as a service (same as *private* outsourcing)
- Blockchain: Ethereum, Starkware etc have deployed SNARGs
- Great Internet Mersenne Prime Search (GIMPS)



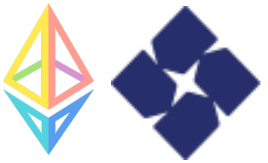
Where are SNARGs Useful?



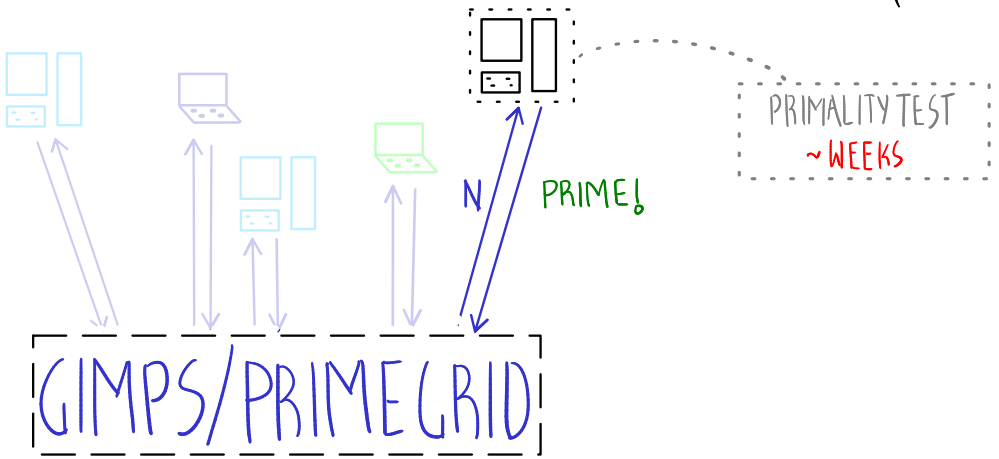
- Compute as a service (same as *private* outsourcing)
- Blockchain: Ethereum, Starkware etc have deployed SNARGs
- Great Internet Mersenne Prime Search (GIMPS)



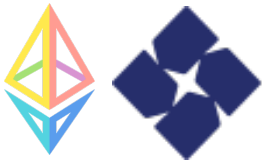
Where are SNARGs Useful?



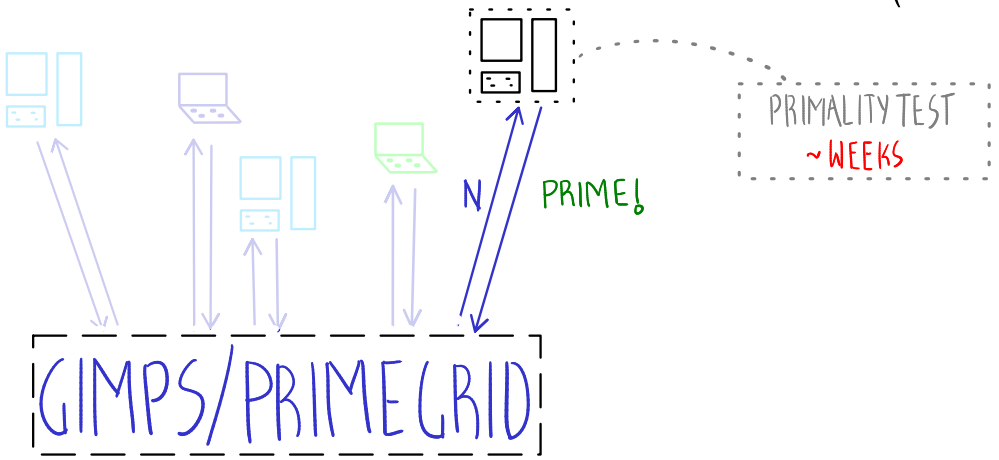
- Compute as a service (same as *private* outsourcing)
- Blockchain: Ethereum, Starkware etc have deployed SNARGs
- Great Internet Mersenne Prime Search (GIMPS)



Where are SNARGs Useful?



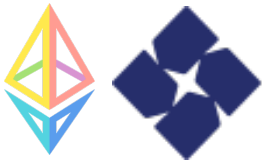
- Compute as a service (same as *private* outsourcing)
- Blockchain: Ethereum, Starkware etc have deployed SNARGs
- Great Internet Mersenne Prime Search (GIMPS)



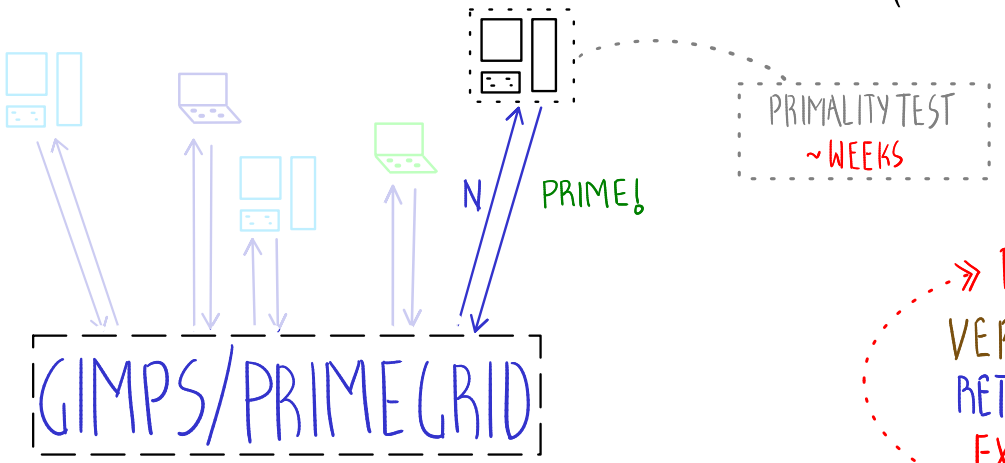
Rank	Number	Discovered	Digits	Form
1	$2^{82589933} - 1$	2018-12-07	24,862,048	Mersenne
2	$2^{77232917} - 1$	2017-12-26	23,249,425	Mersenne
3	$2^{74207281} - 1$	2016-01-07	22,338,618	Mersenne
4	$2^{57885161} - 1$	2013-01-25	17,425,170	Mersenne
5	$2^{43112609} - 1$	2008-08-23	12,978,189	Mersenne
6	$2^{42643801} - 1$	2009-06-04	12,837,064	Mersenne
7	$2^{37156667} - 1$	2008-09-06	11,185,272	Mersenne
8	$2^{32582657} - 1$	2006-09-04	9,808,358	Mersenne
9	$10223 \times 2^{31172165} + 1$	2016-10-31	9,383,761	Proth
10	$2^{30402457} - 1$	2005-12-15	9,152,052	Mersenne

WIKIPEDIA:LARGEST-KNOWN PRIME NUMBER

Where are SNARGs Useful?

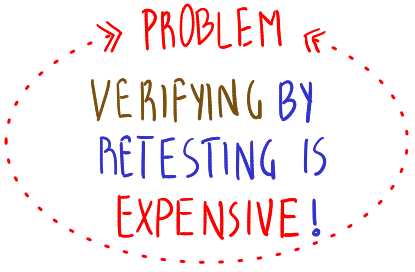


- Compute as a service (same as *private* outsourcing)
- Blockchain: Ethereum, Starkware etc have deployed SNARGs
- Great Internet Mersenne Prime Search (GIMPS)



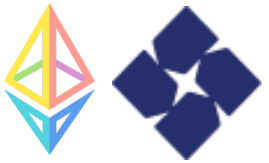
Rank	Number	Discovered	Digits	Form
1	$2^{82589933} - 1$	2018-12-07	24,862,048	Mersenne
2	$2^{77232917} - 1$	2017-12-26	23,249,425	Mersenne
3	$2^{74207281} - 1$	2016-01-07	22,338,618	Mersenne
4	$2^{57885161} - 1$	2013-01-25	17,425,170	Mersenne
5	$2^{43112609} - 1$	2008-08-23	12,978,189	Mersenne
6	$2^{42643801} - 1$	2009-06-04	12,837,064	Mersenne
7	$2^{37156667} - 1$	2008-09-06	11,185,272	Mersenne
8	$2^{32582657} - 1$	2006-09-04	9,808,358	Mersenne
9	$10223 \times 2^{31172165} + 1$	2016-10-31	9,383,761	Proth
10	$2^{30402457} - 1$	2005-12-15	9,152,052	Mersenne

WIKIPEDIA:LARGEST-KNOWN PRIME NUMBER

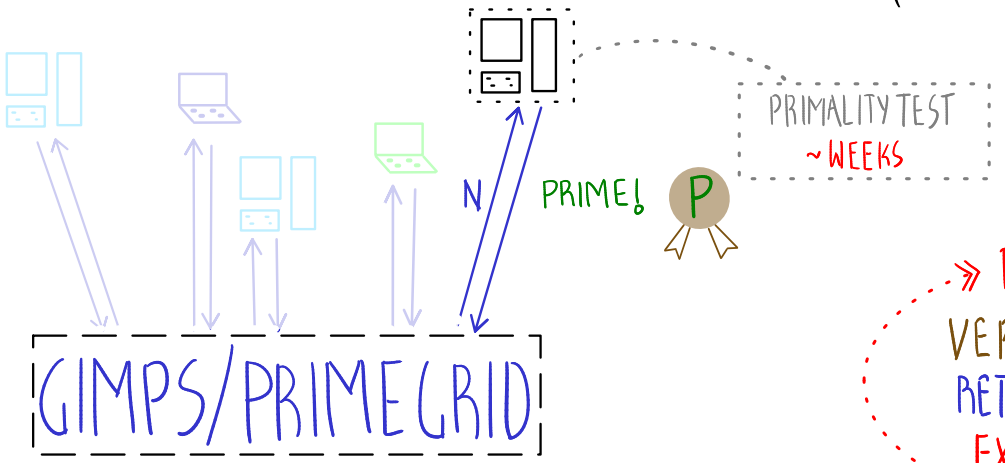


>

Where are SNARGs Useful?

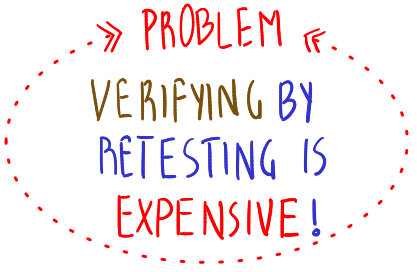


- Compute as a service (same as *private* outsourcing)
- Blockchain: Ethereum, Starkware etc have deployed SNARGs
- Great Internet Mersenne Prime Search (GIMPS)



Rank	Number	Discovered	Digits	Form
1	$2^{82589933} - 1$	2018-12-07	24,862,048	Mersenne
2	$2^{77232917} - 1$	2017-12-26	23,249,425	Mersenne
3	$2^{74207281} - 1$	2016-01-07	22,338,618	Mersenne
4	$2^{57885161} - 1$	2013-01-25	17,425,170	Mersenne
5	$2^{43112609} - 1$	2008-08-23	12,978,189	Mersenne
6	$2^{42643801} - 1$	2009-06-04	12,837,064	Mersenne
7	$2^{37156667} - 1$	2008-09-06	11,185,272	Mersenne
8	$2^{32582657} - 1$	2006-09-04	9,808,358	Mersenne
9	$10223 \times 2^{31172165} + 1$	2016-10-31	9,383,761	Proth
10	$2^{30402457} - 1$	2005-12-15	9,152,052	Mersenne

WIKIPEDIA:LARGEST-KNOWN PRIME NUMBER



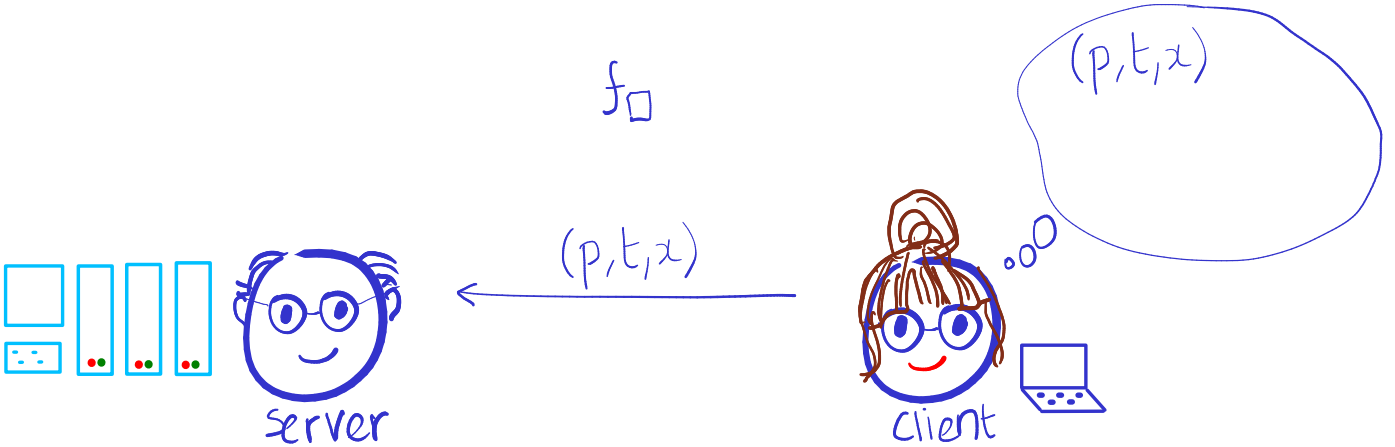
>

Plan for this Session

1. Verifiable Outsourcing of Computation
   - Succinct Non-Interactive Argument (SNARG)

2. SNARG for Repeated Squaring
   - Step I: Interactive Proof for Repeated Squaring
   - Step II: Remove Interaction via Fiat-Shamir Transform

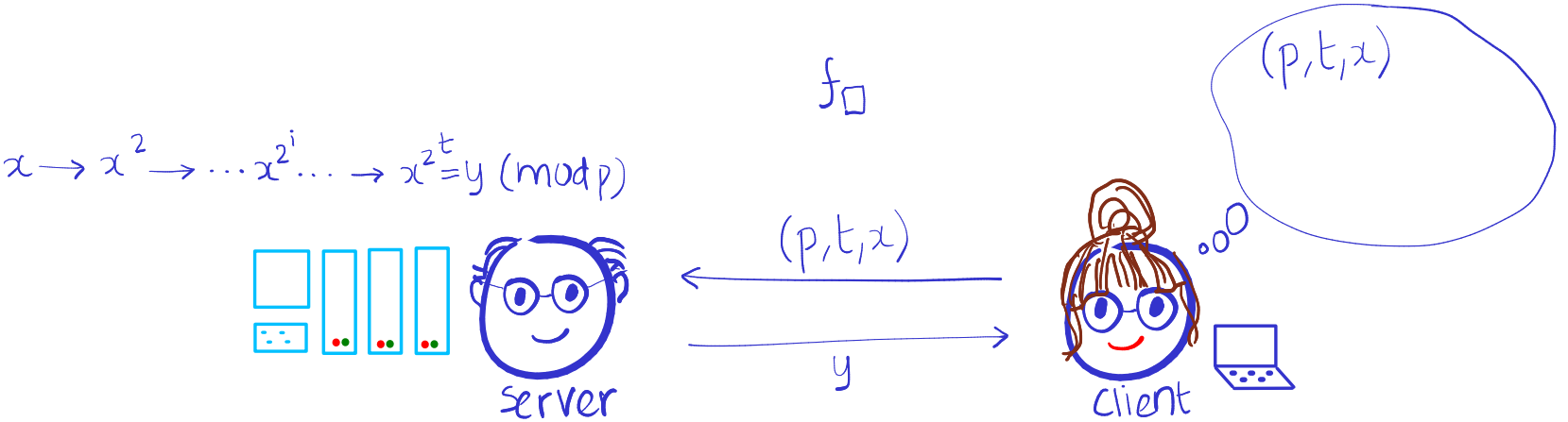
Repeated Squaring Function over RSA Group...



- Repeated squaring function modulo prime p:

$$f_\square(p, t, x) := x^{2^t} \bmod p$$

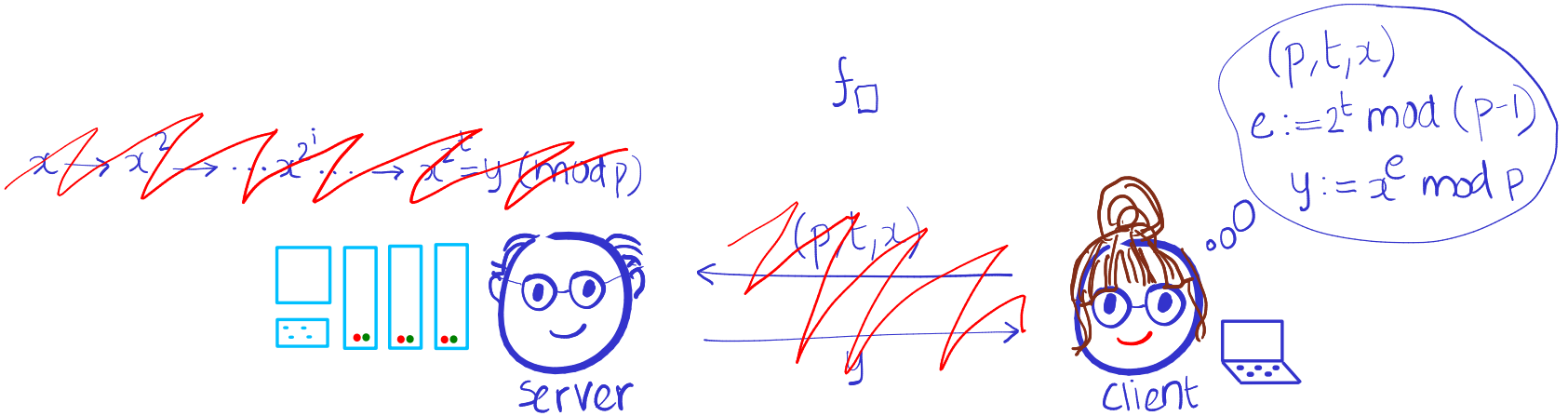
Repeated Squaring Function over RSA Group...



- Repeated squaring function modulo prime p:

$$f_\square(p, t, x) := x^{2^t} \bmod p$$

Repeated Squaring Function over RSA Group

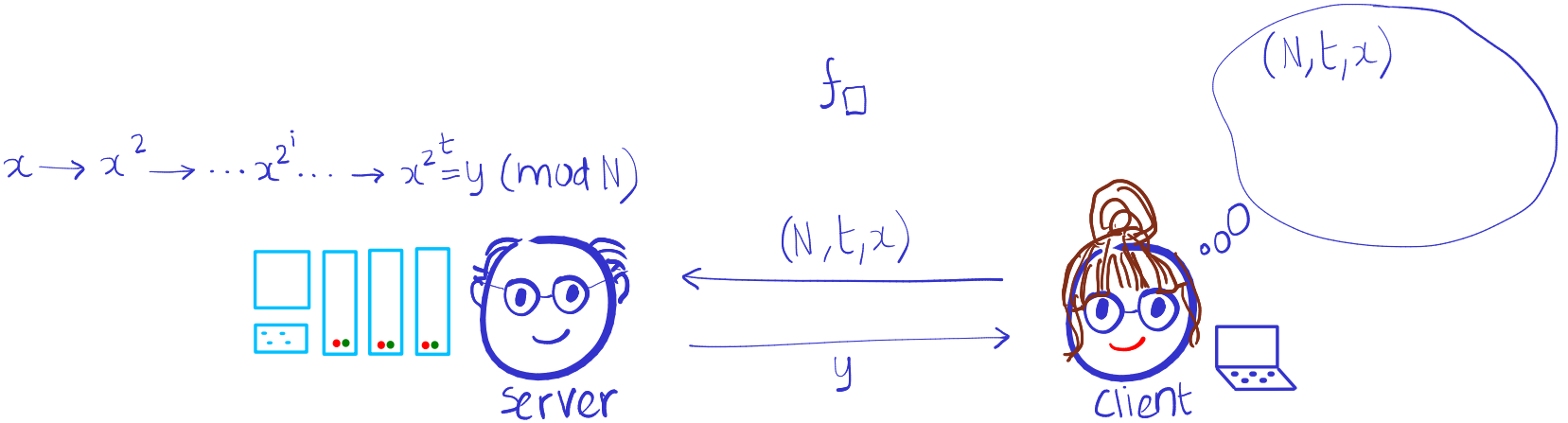


- Repeated squaring function modulo prime p:

$$f_\square(p, t, x) := x^{2^t} \bmod p$$

  - Shortcut: 1) $e := 2^t \bmod (p-1)$ 2) $y := x^e \bmod p$
  - Requires $\approx n \log(t)$ multiplications (n: $\hookrightarrow |p|$)

Repeated Squaring Function over RSA Group



- Repeated squaring function modulo prime p:

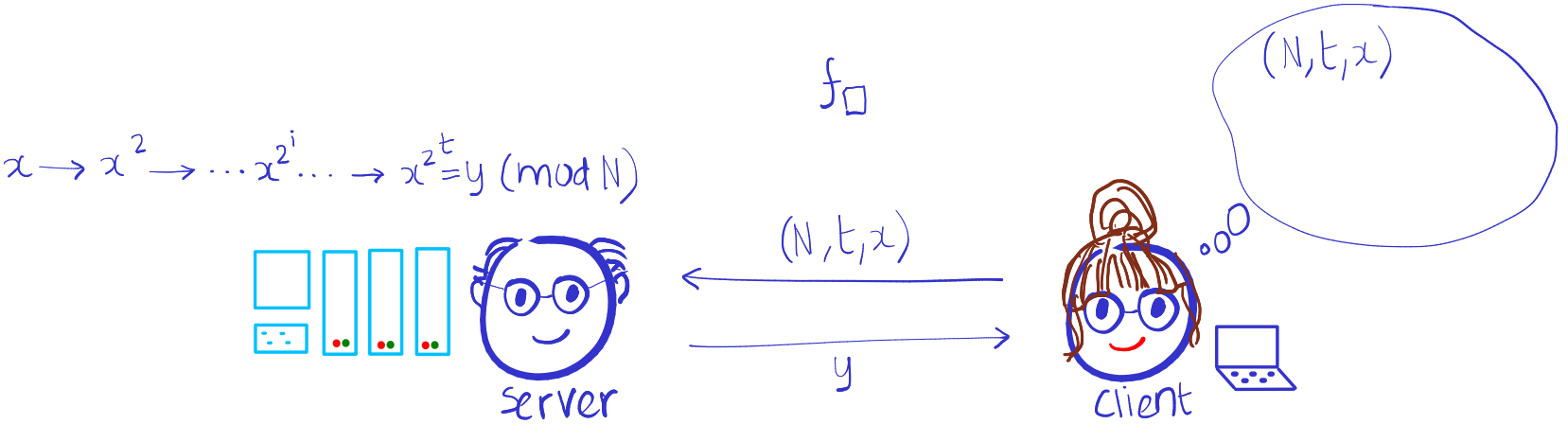
$$f_\square(p, t, x) := x^{2^t} \bmod p$$

  - Shortcut: 1) $e := 2^t \bmod (p-1)$ 2) $y := x^e \bmod p$
  - Requires $\approx n \log(t)$ multiplications (n: $|p|$)

- Repeated squaring function modulo *composite* $N = pq$

$$f_\square(N, t, x) := x^{2^t} \bmod N$$

Repeated Squaring Function over RSA Group



- Repeated squaring function modulo prime p:

$$f_\square(p, t, x) := x^{2^t} \bmod p$$

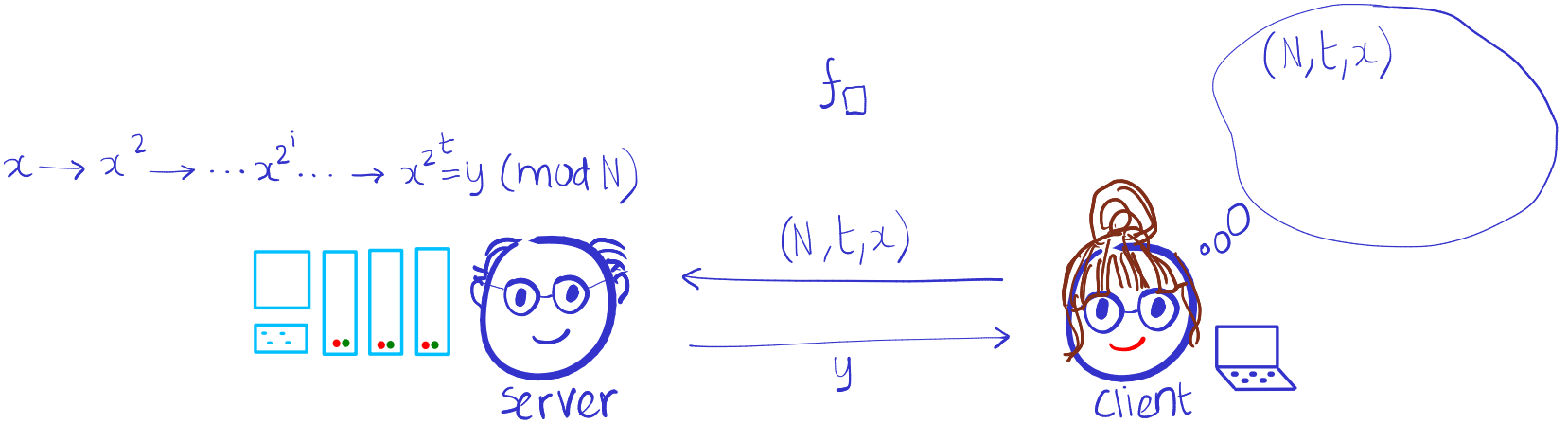
  - Shortcut: 1) $e := 2^t \bmod (p-1)$ 2) $y := x^e \bmod p$
  - Requires $\approx n \log(t)$ multiplications ($\hookrightarrow |p|$)

- Repeated squaring function modulo *composite* $N = pq$

$$f_\square(N, t, x) := x^{2^t} \bmod N$$

  - Shortcut: 1) $e := 2^t \bmod (p-1)(q-1)$ 2) $y := x^e \bmod N$

Repeated Squaring Function over RSA Group



- Repeated squaring function modulo prime p:

$$f_\square(p, t, x) := x^{2^t} \bmod p$$

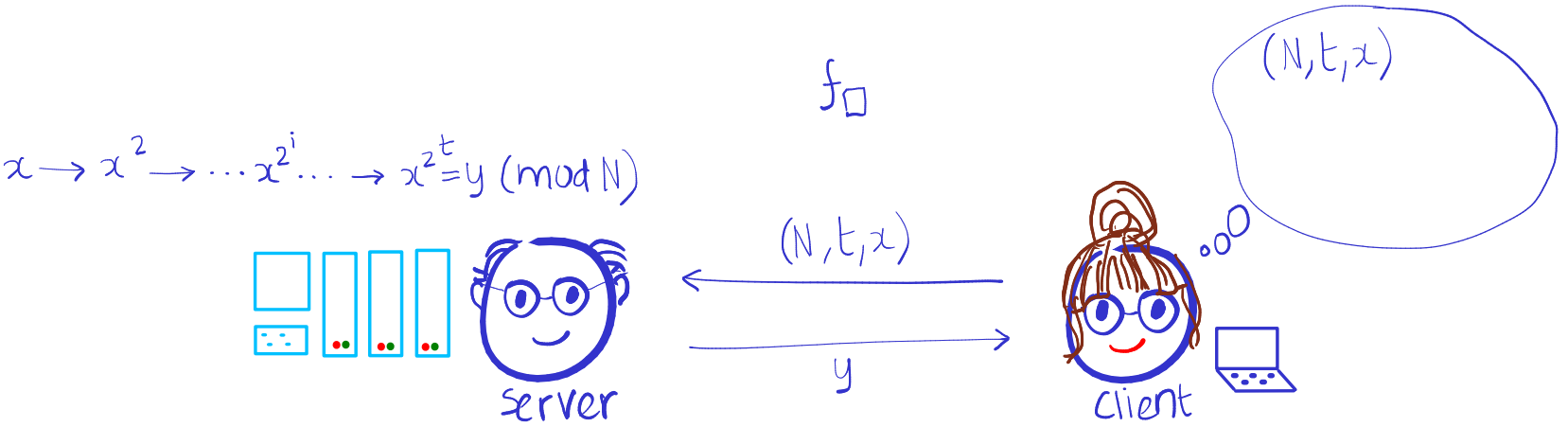
  - Shortcut: 1) $e := 2^t \bmod (p-1)$ 2) $y := x^e \bmod p$
  - Requires $\approx n \log(t)$ multiplications ($\hookrightarrow |p|$)

- Repeated squaring function modulo *composite* $N = pq$

$$f_\square(N, t, x) := x^{2^t} \bmod N$$

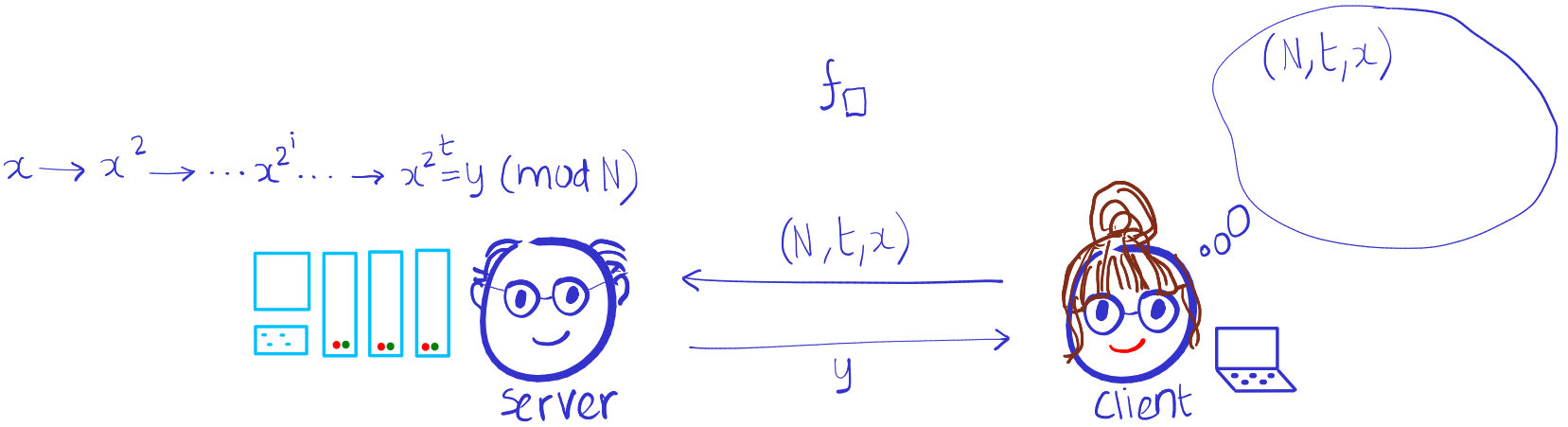
  - Shortcut: 1) $e := 2^t \bmod (p-1)(q-1)$ 2) $y := x^e \bmod N$
  - Requires $\approx n \log(t)$ multiplications *if factors of N known* ($\hookrightarrow |N|$)

Repeated Squaring Function over RSA Group...



- What if factors of N are *not known*?
  - Server could factor N and return (p, q)

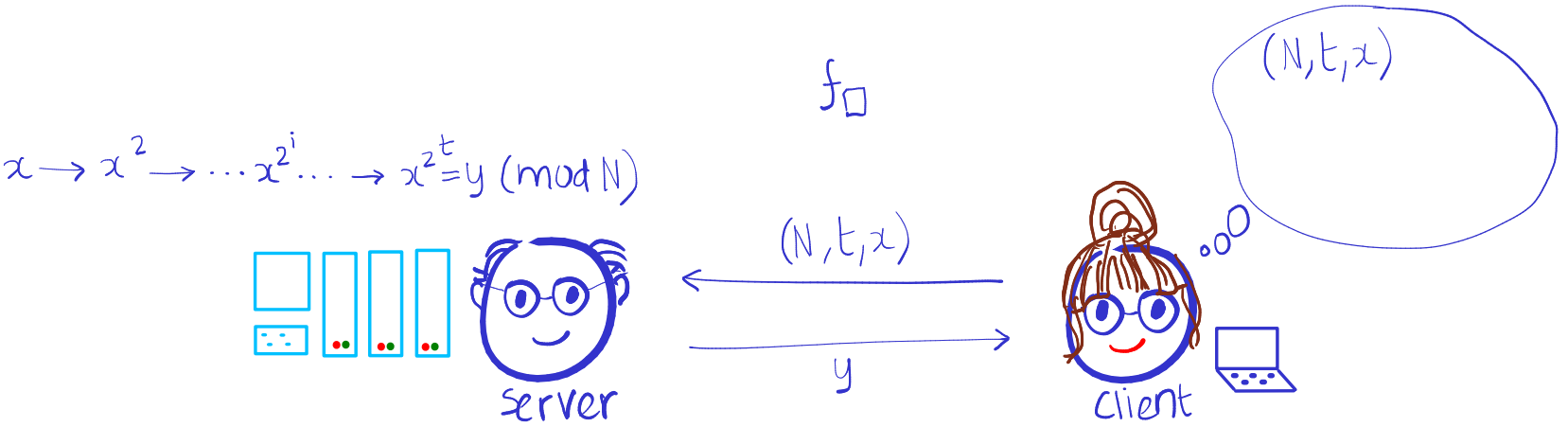
Repeated Squaring Function over RSA Group...



- What if factors of N are *not known*?
  - Server could factor N and return (p, q)

⚠ Recall: we need honest prover to not have too much overhead

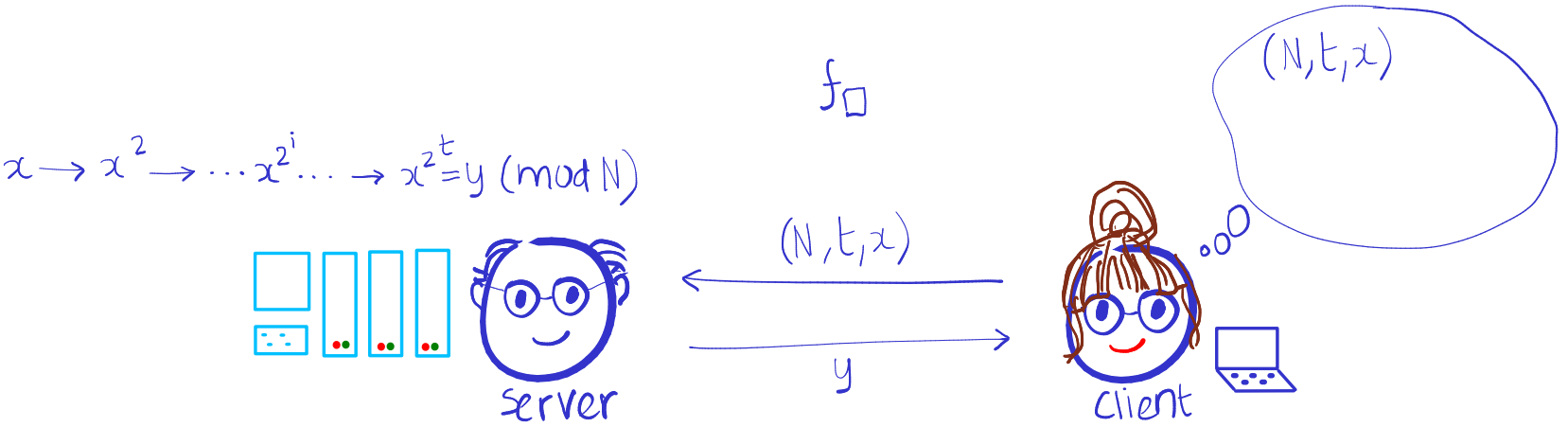
Repeated Squaring Function over RSA Group...



- What if factors of N are *not known*?
  - Server could factor N and return (p, q)

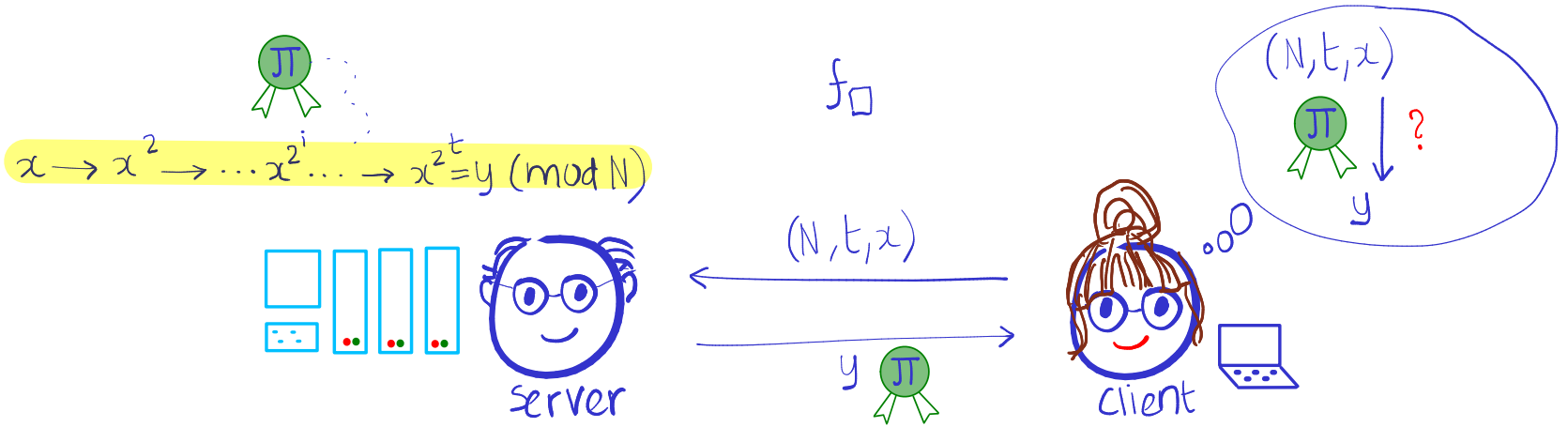
  ⚠ Recall: we need honest prover to not have too much overhead
- Without factors, fastest way to compute $f_\square(N, t, x)$ believed to be by repeated squaring

Repeated Squaring Function over RSA Group...



- What if factors of N are *not known*?
  - Server could factor N and return (p, q)
  - ⚠ Recall: we need honest prover to not have too much overhead
- Without factors, fastest way to compute $f_\square(N, t, x)$ believed to be by repeated squaring
  - Even if server has $\mathbf{poly}(n)$ amount of parallelism!
  - So-call RSW assumption: useful in "timed" cryptography
    - Time-lock puzzles
    - Verifiable delay functions

Repeated Squaring Function over RSA Group...



- What if factors of N are *not known*?
  - Server could factor N and return (p, q)
  - ⚠ Recall: we need honest prover to not have too much overhead
- Without factors, fastest way to compute $f_\square(N, t, x)$ believed to be by repeated squaring
  - Even if server has $\mathbf{poly}(n)$ amount of parallelism!
  - So-call RSW assumption: useful in "timed" cryptography
    - Time-lock puzzles
    - Verifiable delay functions
- A proof π would be useful here

Plan for this Session

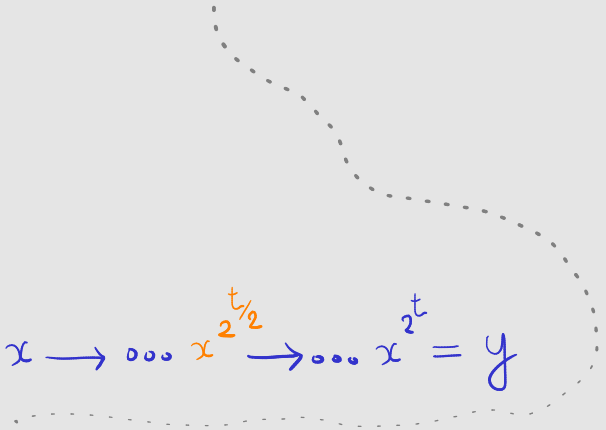
1. Verifiable Outsourcing of Computation
   - Succinct Non-Interactive Argument (SNARG)

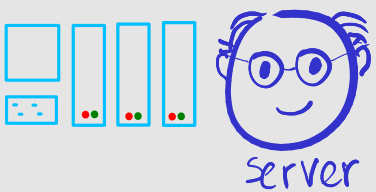
2. SNARG for Repeated Squaring
   - Step I: Interactive Proof for Repeated Squaring
   - Step II: Remove Interaction via Fiat-Shamir Transform

Step I: Harness the Power of IP

Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)

$t = 2^a$

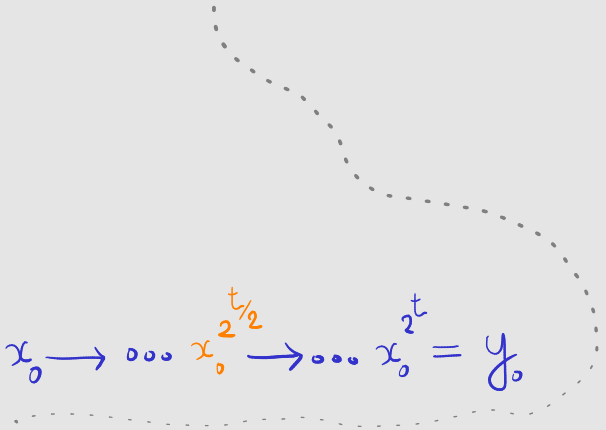


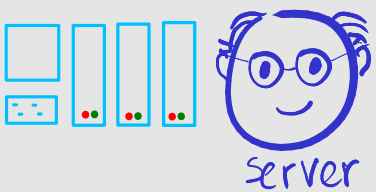


Step I: Harness the Power of IP

Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)

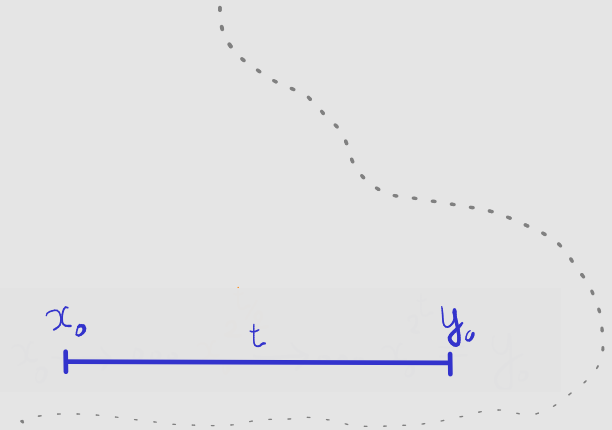
$t = 2^d$

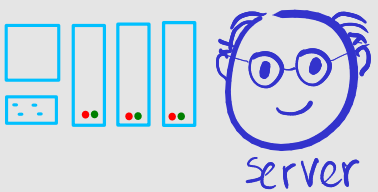




Step I: Harness the Power of IP

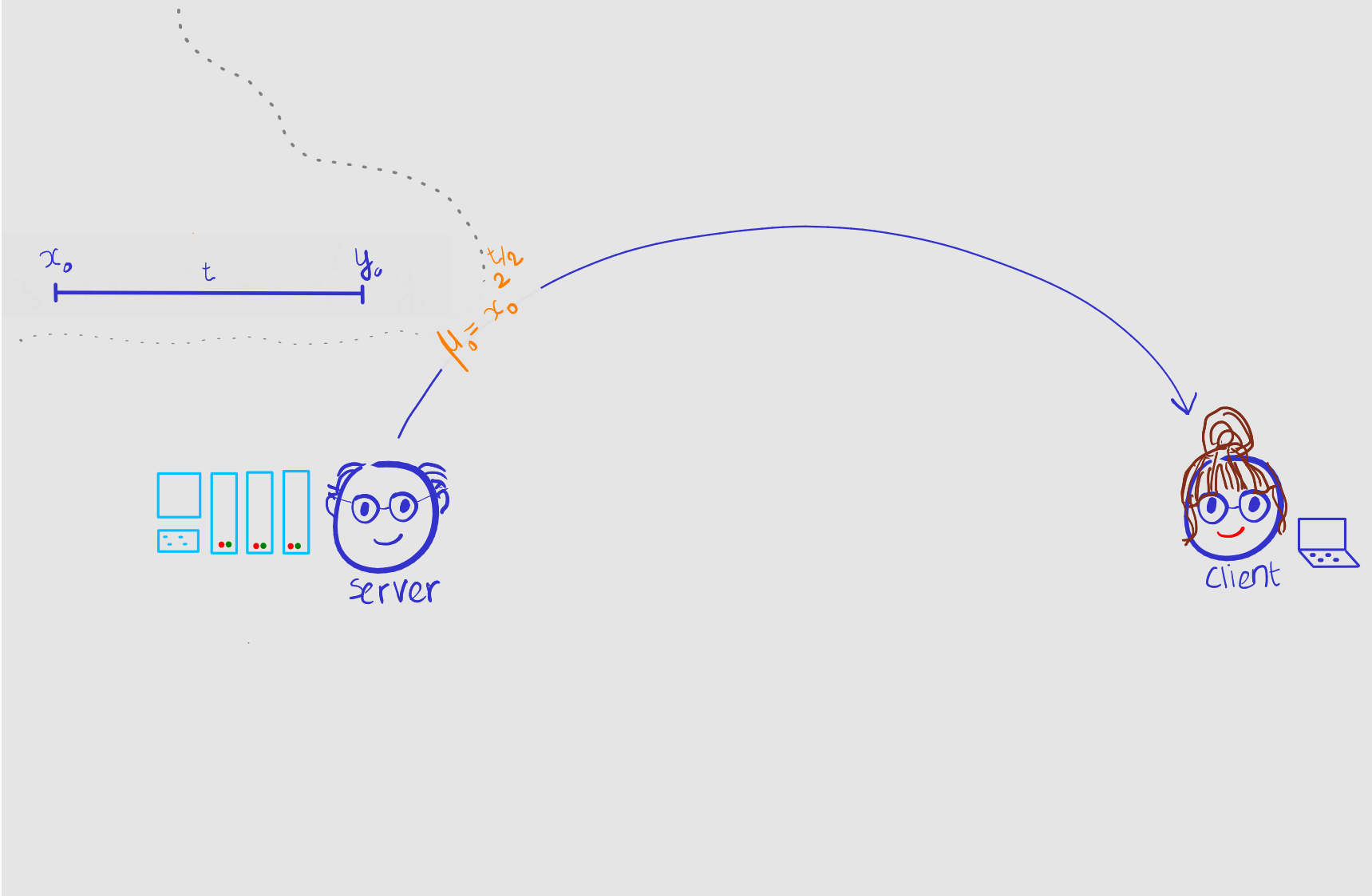
Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)





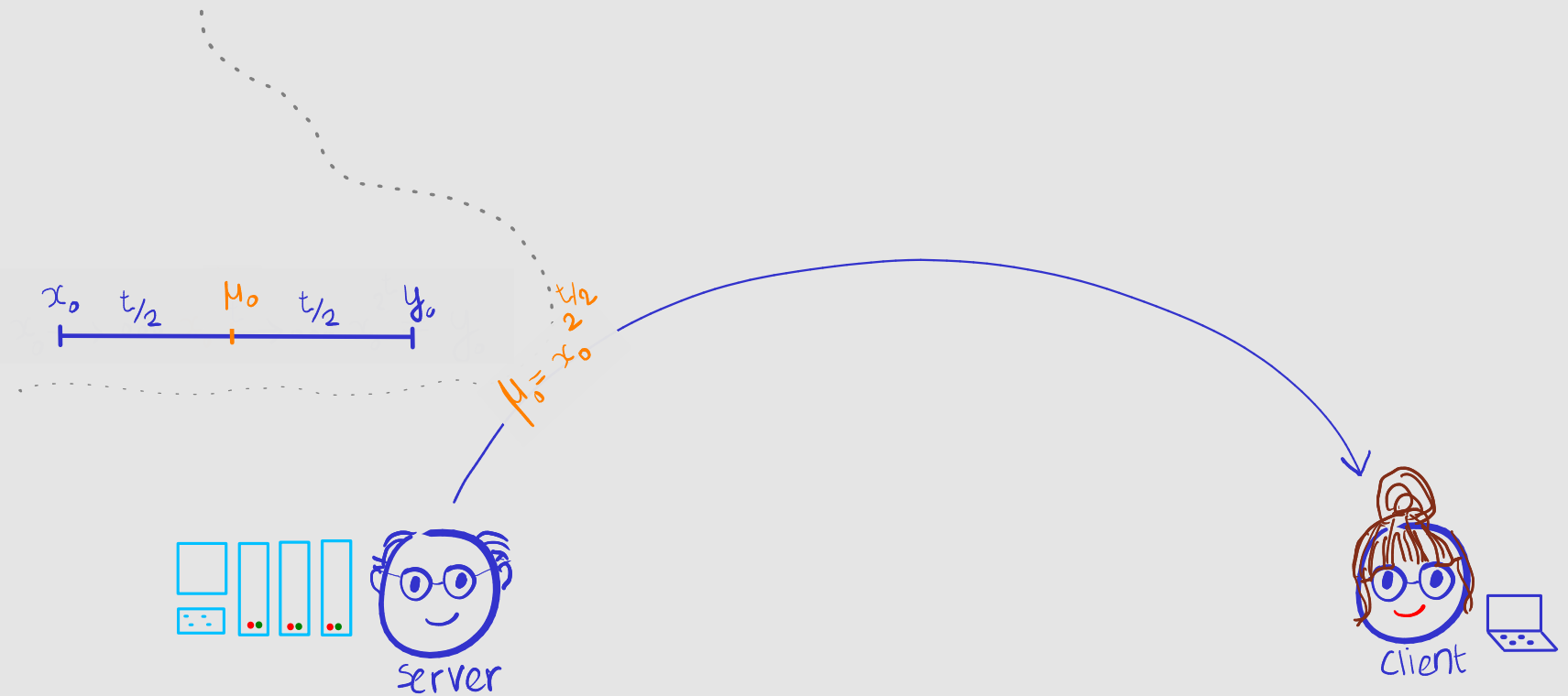
Step I: Harness the Power of IP

Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)



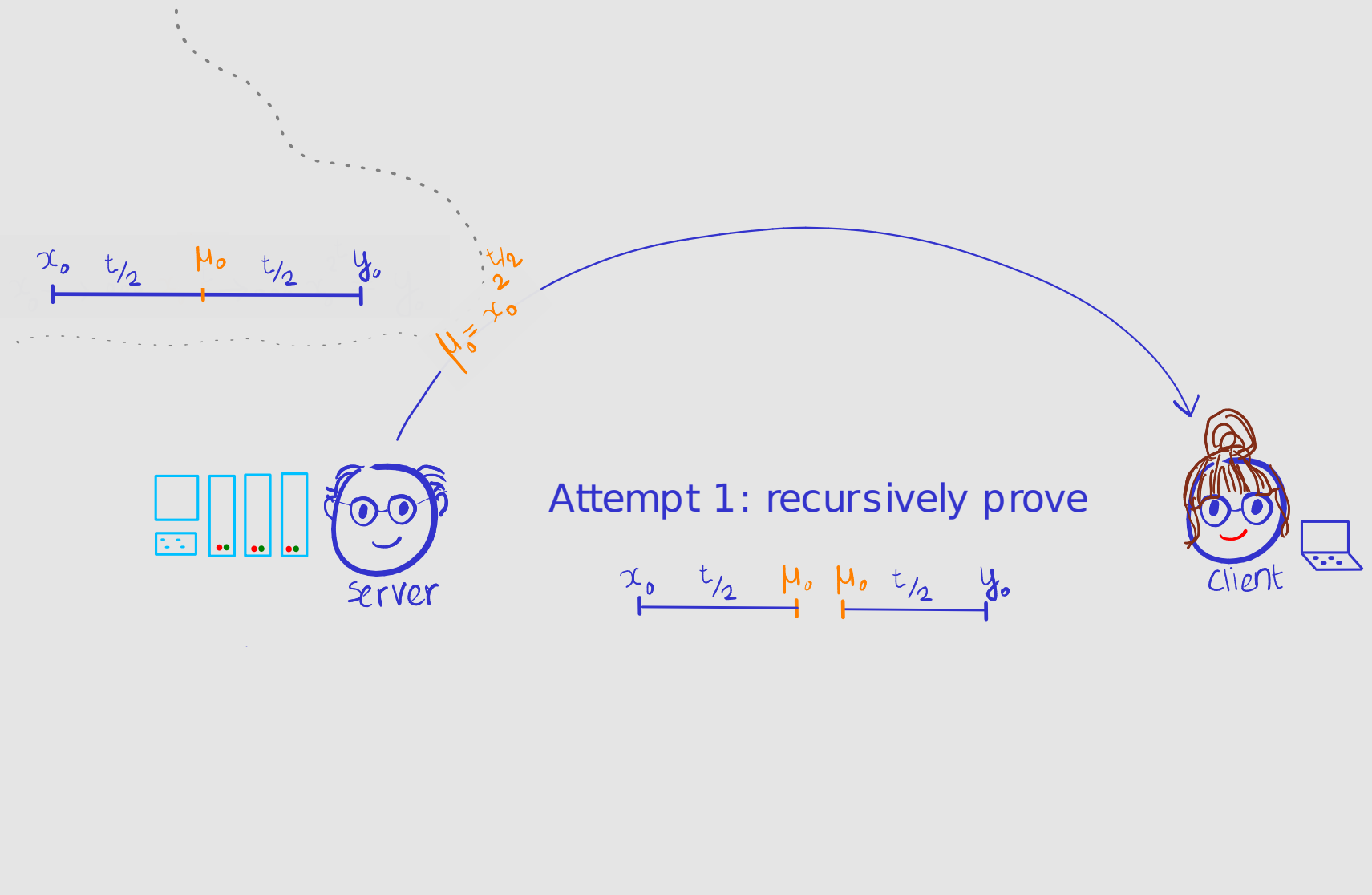
Step I: Harness the Power of IP

Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)



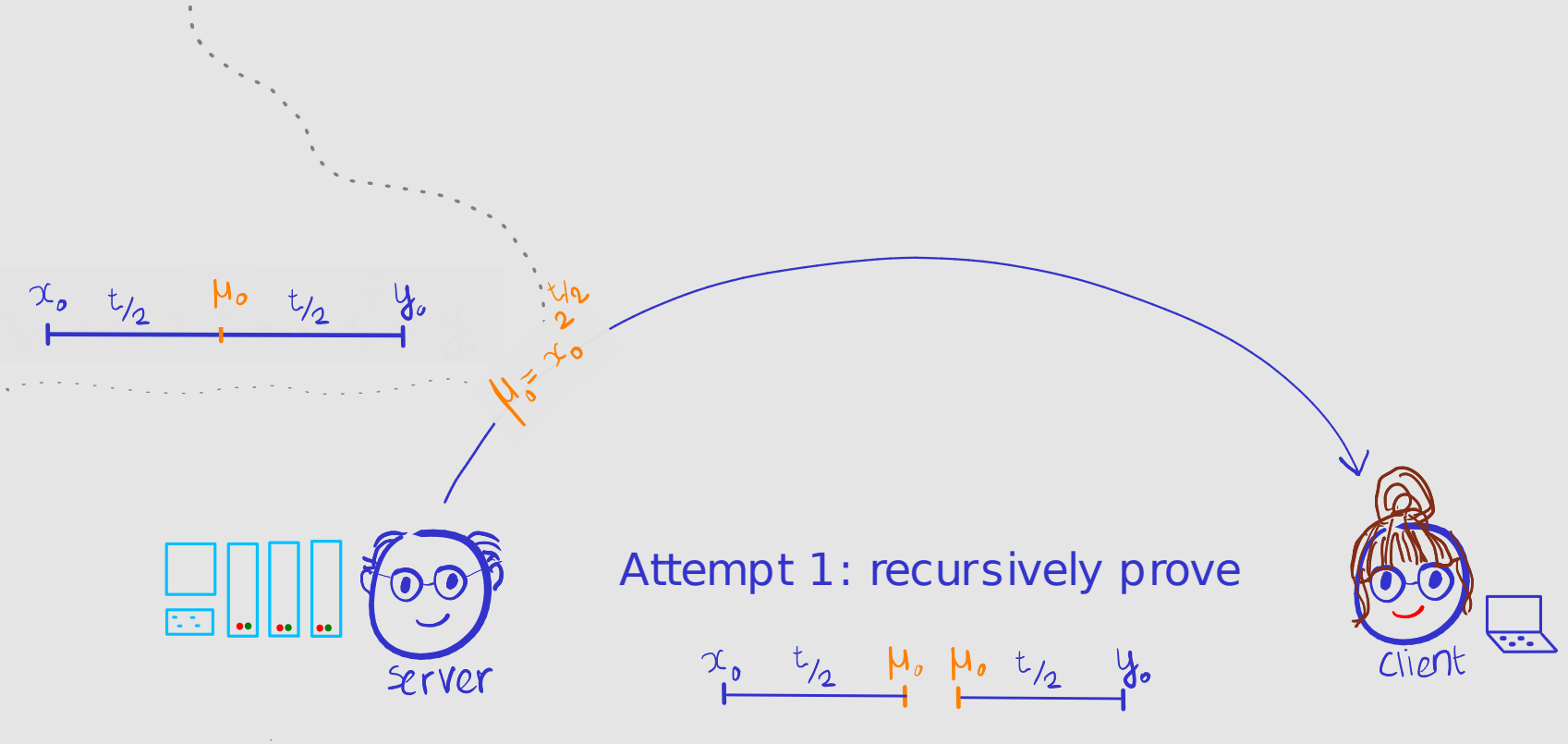
Step I: Harness the Power of IP

Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)



Step I: Harness the Power of IP

Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)

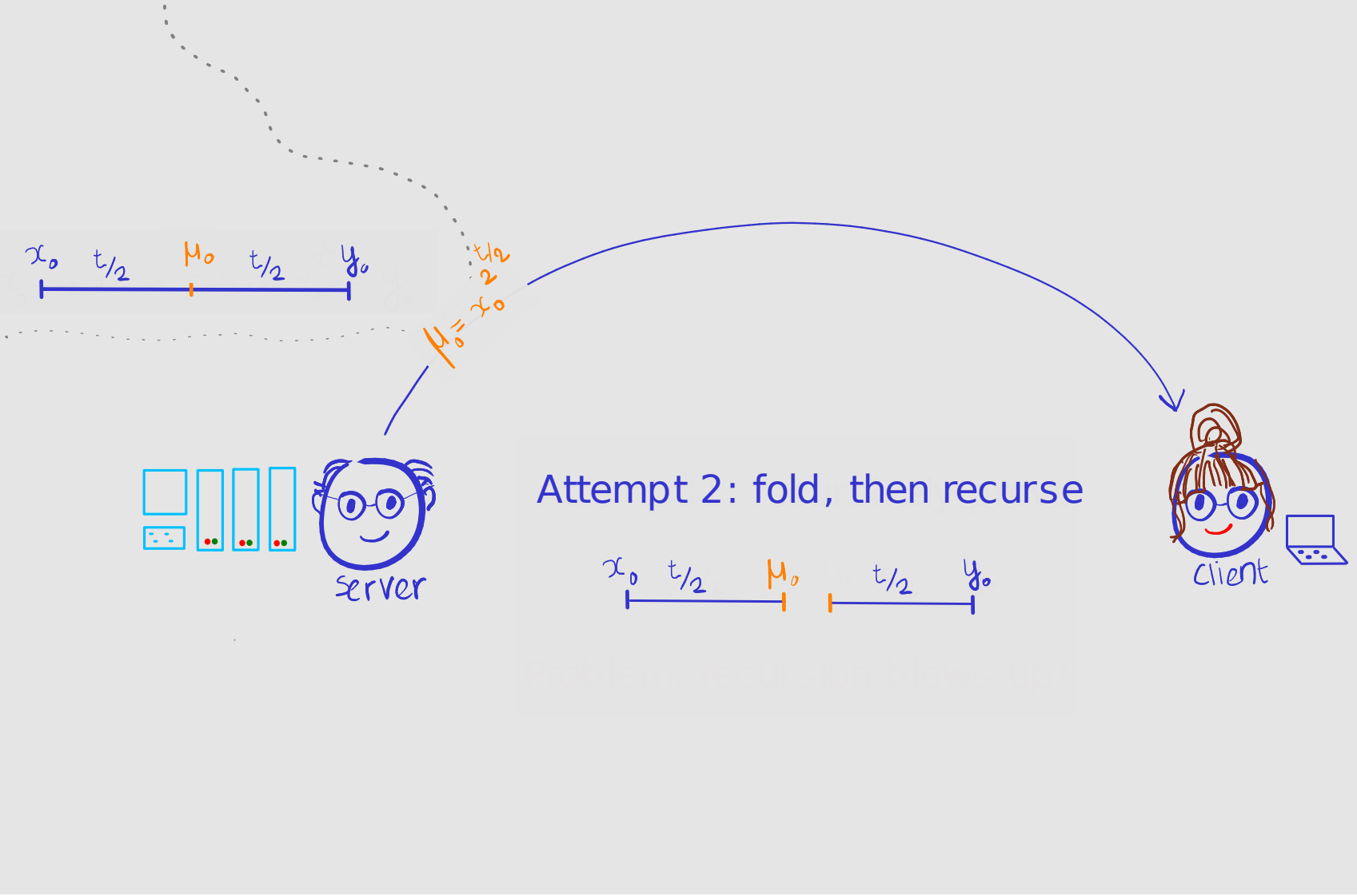


Attempt 1: recursively prove

Problem: recursion blows up!

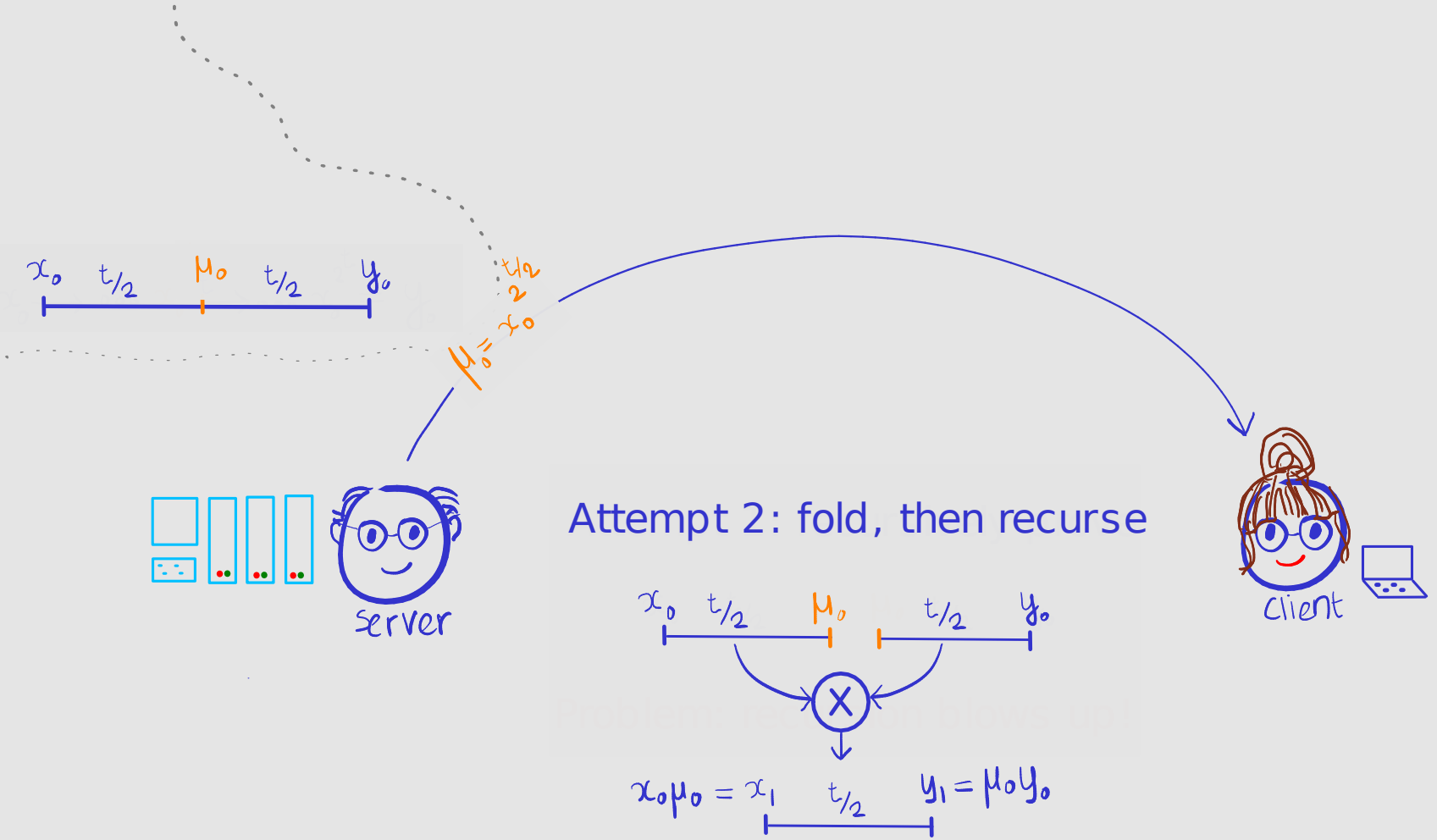
Step I: Harness the Power of IP

Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)



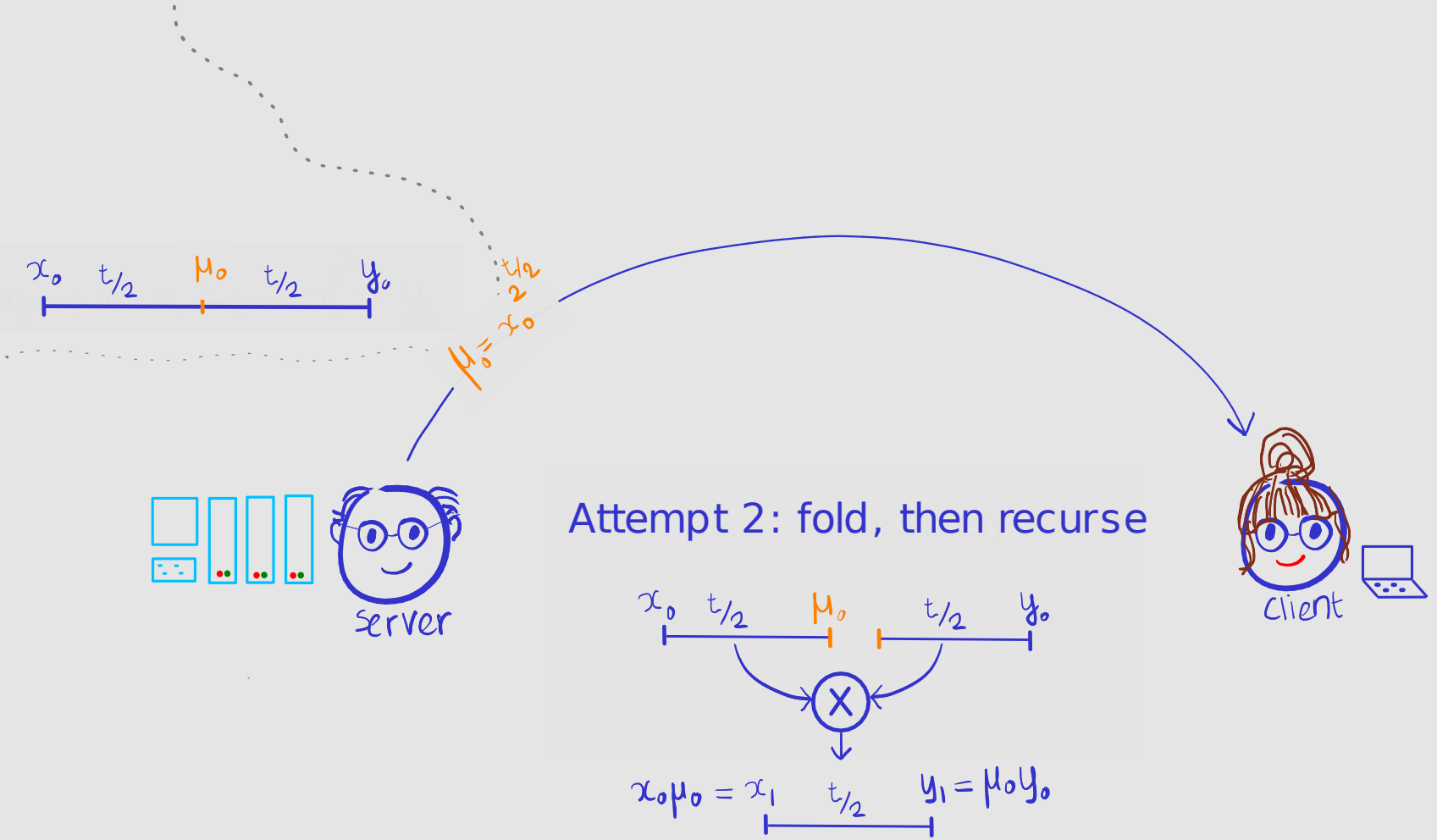
Step I: Harness the Power of IP

Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)



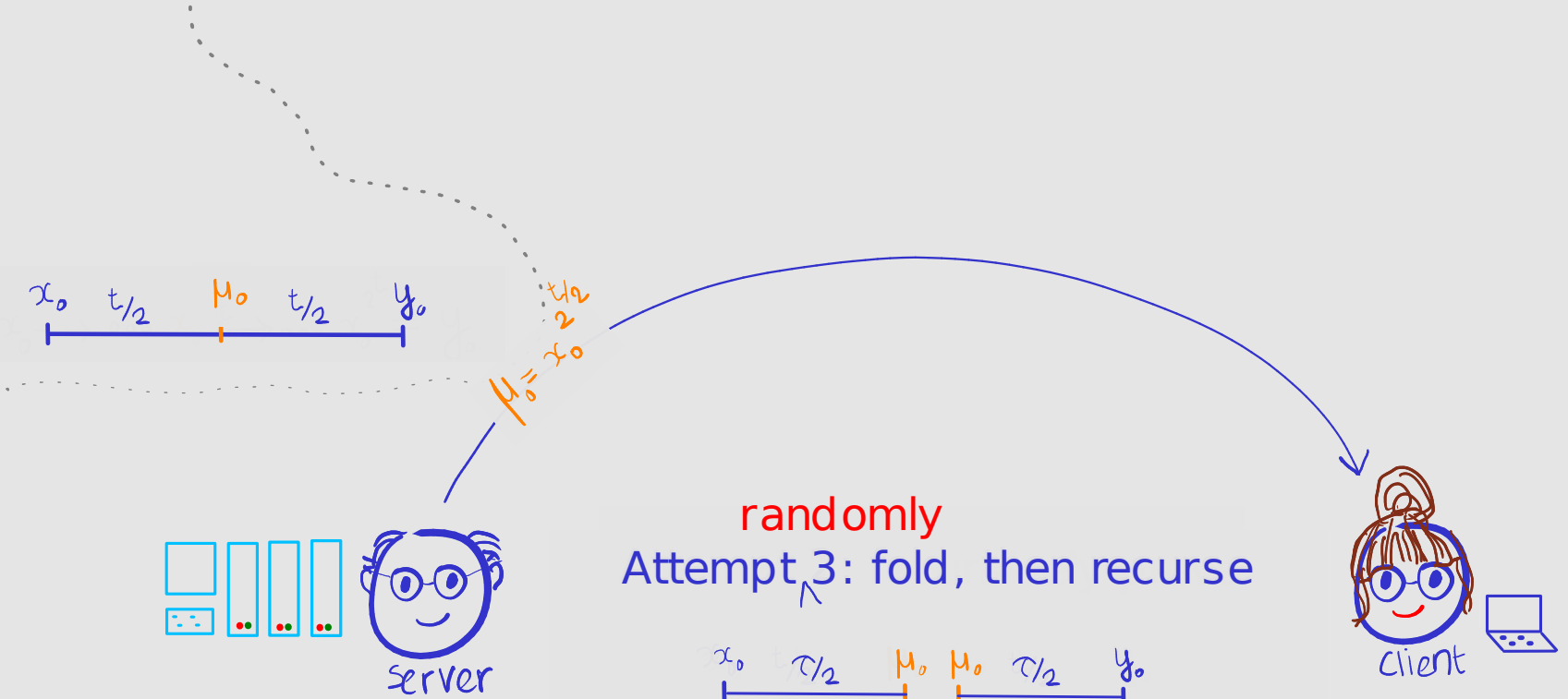
Step I: Harness the Power of IP

Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)



Step I: Harness the Power of IP

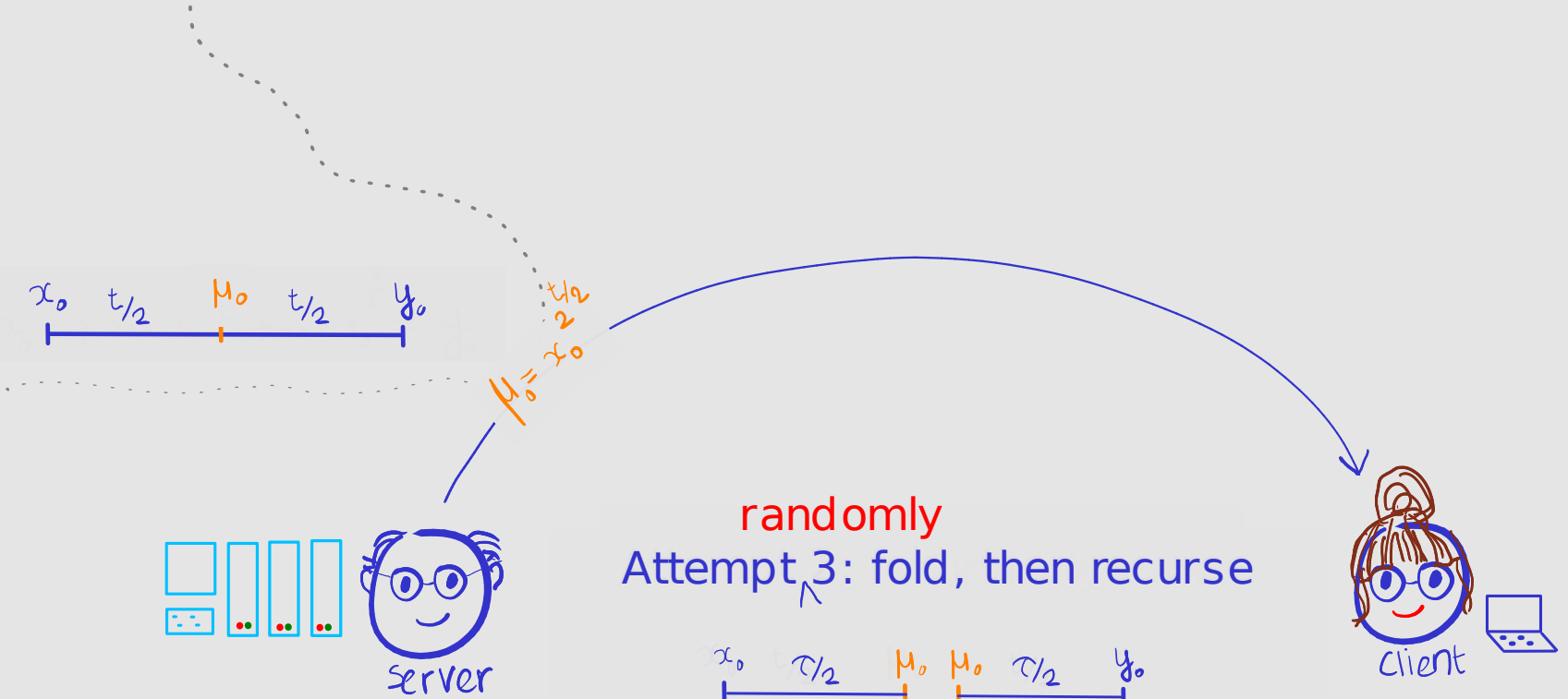
Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)



Problem: Bob can cheat. How?

Step I: Harness the Power of IP

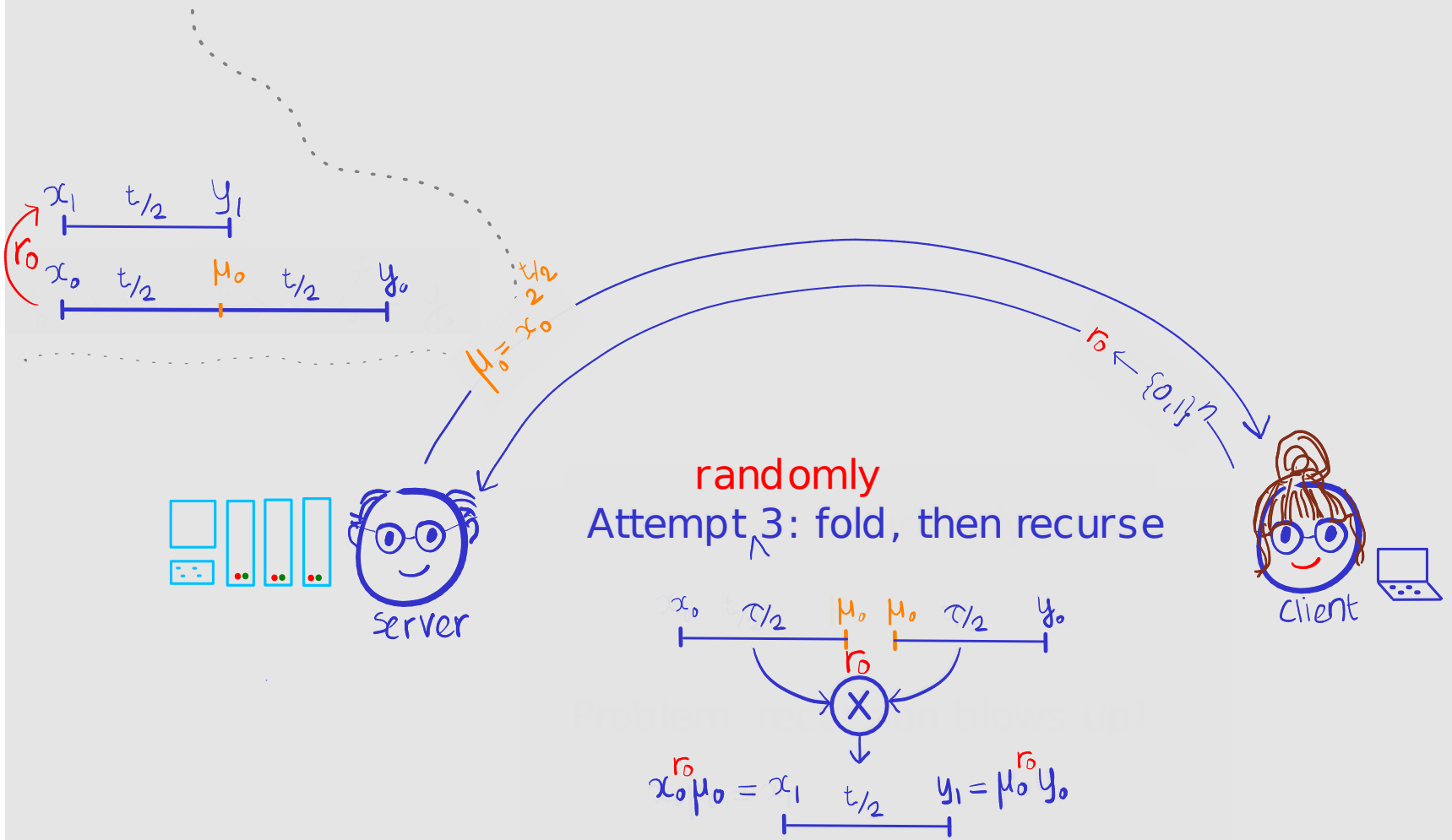
Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)



Problem: Bob can cheat. How?

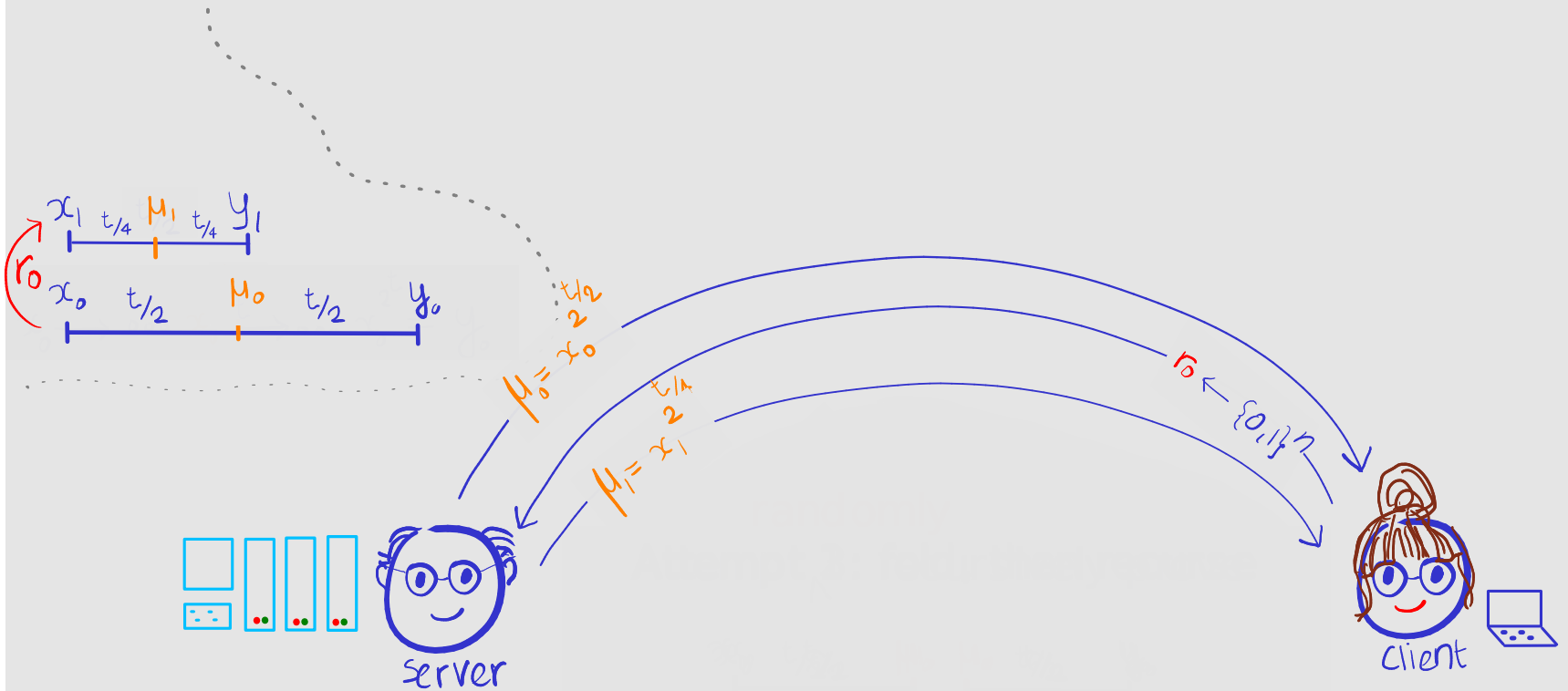
Step I: Harness the Power of IP

Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)



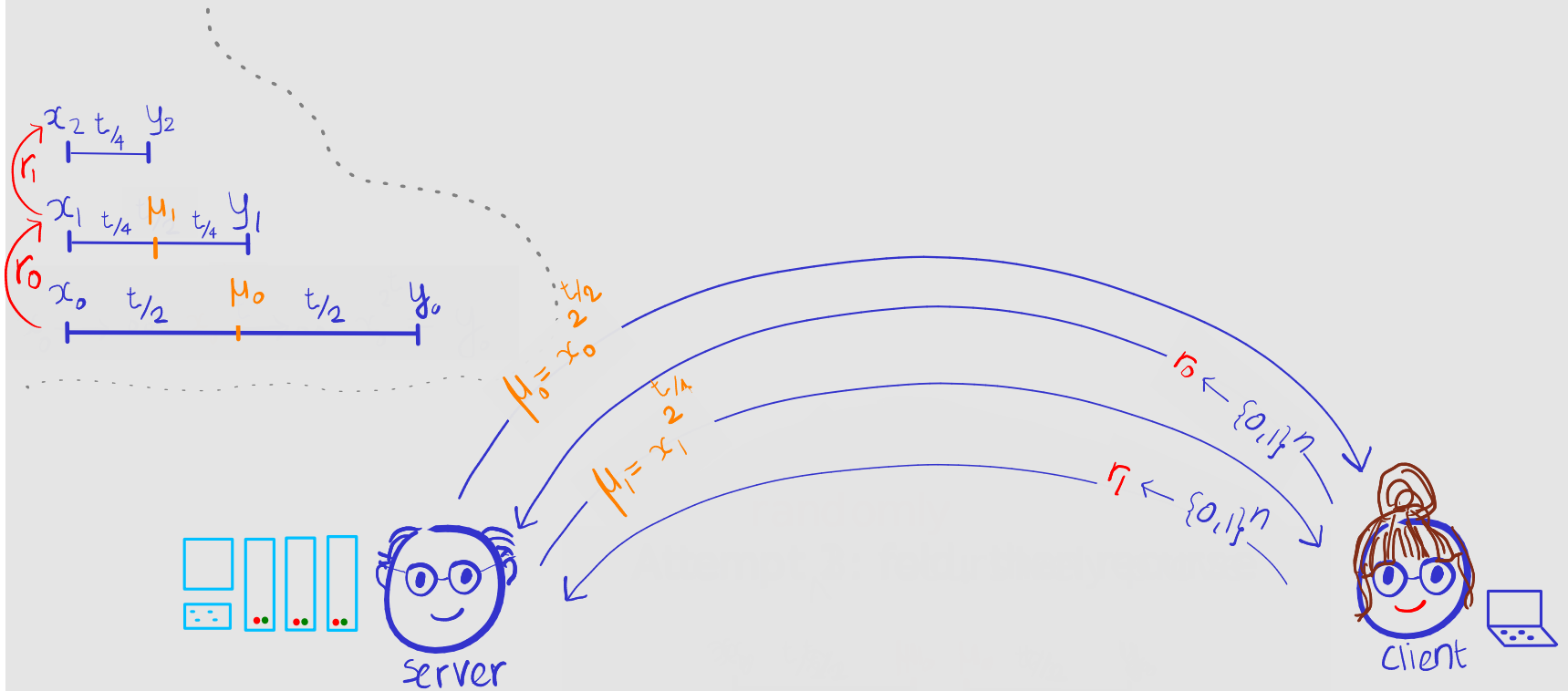
Step I: Harness the Power of IP

Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)



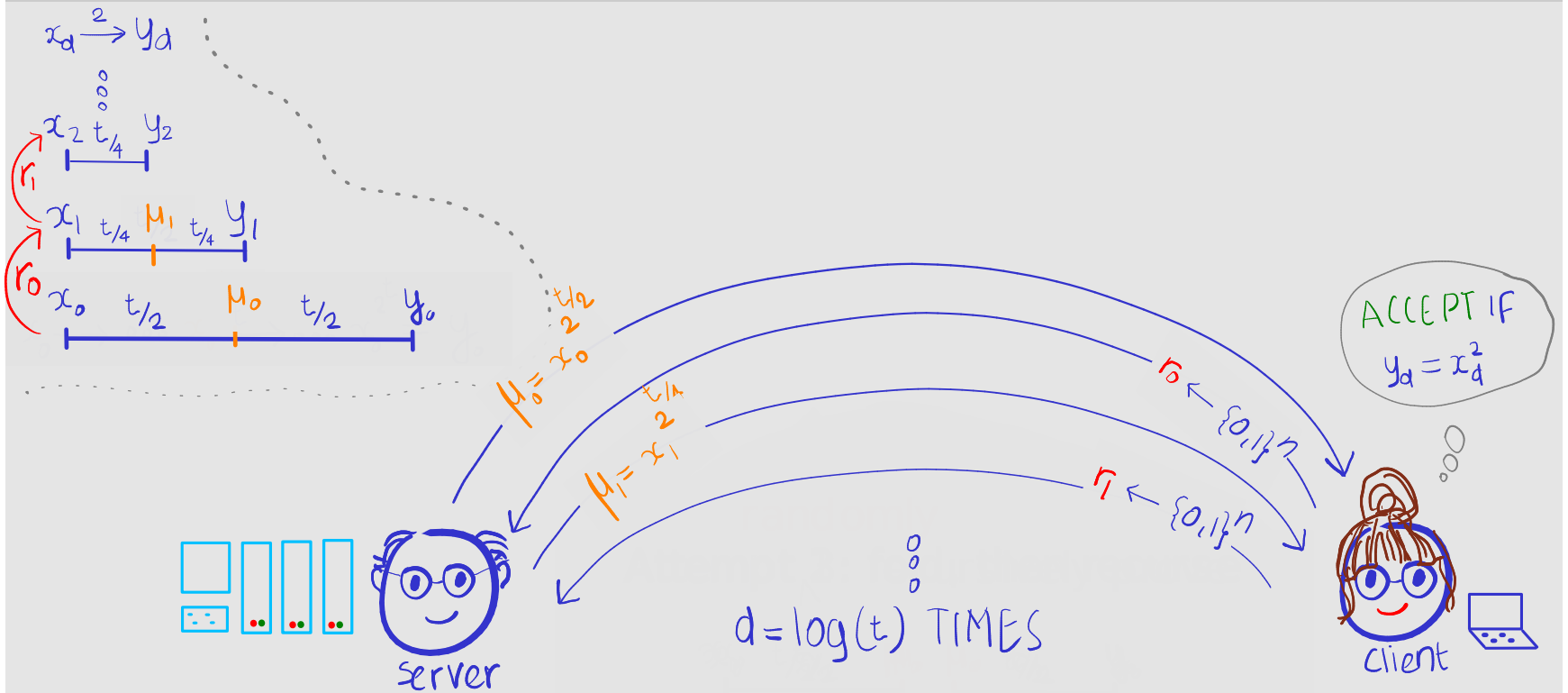
Step I: Harness the Power of IP

Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)



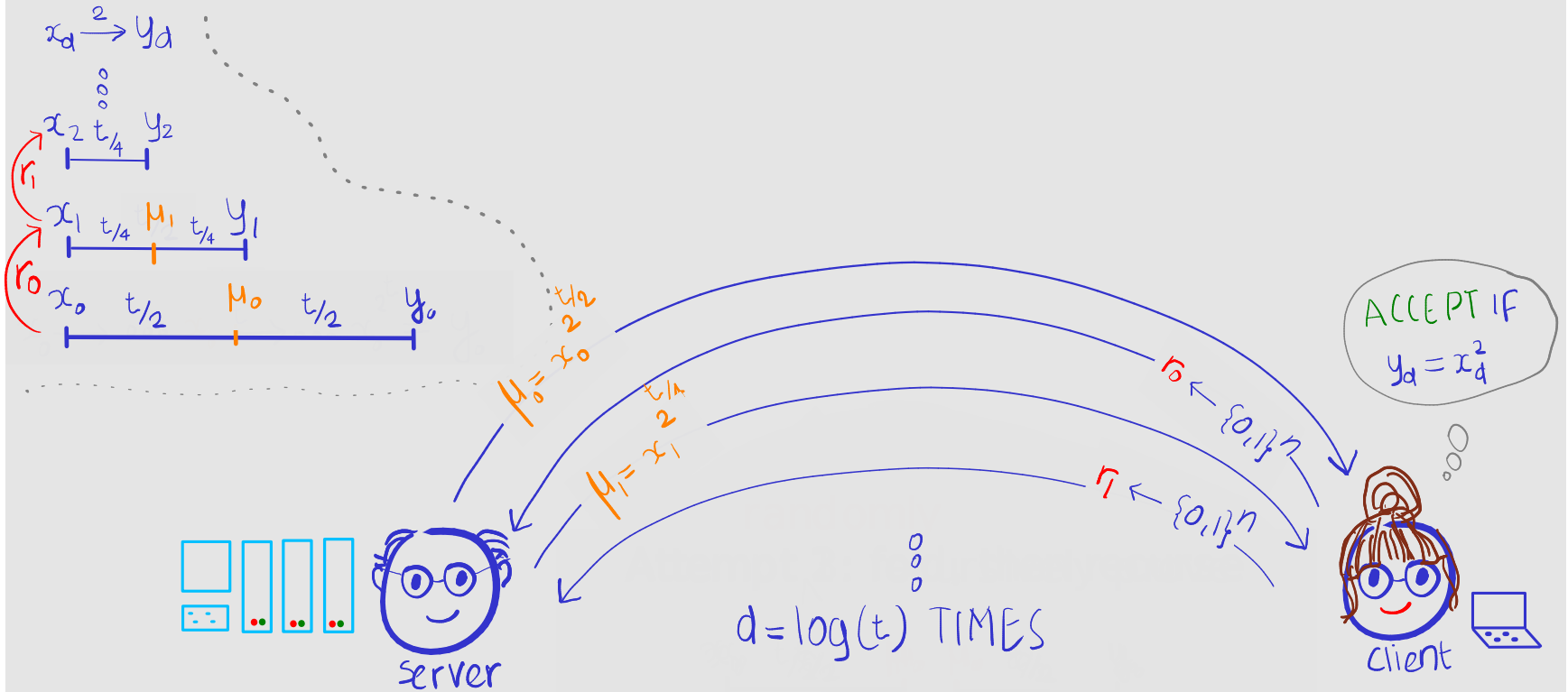
Step I: Harness the Power of IP

Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)



Step I: Harness the Power of IP

Protocol 1 (Π, proves that $y = x^{2^t} \bmod N$ interactively)



? Why is the protocol complete?

Why is Π Sound?

Theorem 1

Π *is an interactive proof for the language*

$$\mathcal{L}_{\mathsf{f}_\square} := \left\{ (x, y, t, N) : y = x^{2^t} \bmod N \right\}$$

Why is Π Sound?

Theorem 1

Π *is an interactive proof for the language*

$$\mathcal{L}_{\mathsf{f}_\square} := \left\{ (x, y, t, N) : y = x^{2^t} \bmod N \right\}$$

Proof Sketch.

Statistical soundness (for first round) can be argued using:

1. Claim: if $y_0 \neq x_0^{2^t}$ then $y_0 \neq \mu_0^{2^{t/2}}$ *or* $\mu_0 \neq x_0^{2^{t/2}}$ must hold ?

Why is Π Sound?

Theorem 1

Π *is an interactive proof for the language*

$$\mathcal{L}_{\mathsf{f}_\square} := \left\{ (x, y, t, N) : y = x^{2^t} \bmod N \right\}$$

Proof Sketch.

Statistical soundness (for first round) can be argued using:

1. Claim: if $y_0 \neq x_0^{2^t}$ then $y_0 \neq \mu_0^{2^{t/2}}$ *or* $\mu_0 \neq x_0^{2^{t/2}}$ must hold ?

$(x_0, y_0, t) \notin \mathcal{L}_{\mathsf{f}_\square}$

$(x_0, \mu_0, t/2) \notin \mathcal{L}_{\mathsf{f}_\square}$

$(\mu_0, y_0, t/2) \notin \mathcal{L}_{\mathsf{f}_\square}$

Why is Π Sound?

Theorem 1

Π *is an interactive proof for the language*

$$\mathcal{L}_{f_\square} := \left\{ (x, y, t, N) : y = x^{2^t} \bmod N \right\}$$

Proof Sketch.

Statistical soundness (for first round) can be argued using:

1. Claim: if $y_0 \neq x_0^{2^t}$ then $y_0 \neq \mu_0^{2^{t/2}}$ *or* $\mu_0 \neq x_0^{2^{t/2}}$ must hold
2. Claim: if $y_0 \neq \mu_0^{2^{t/2}}$ *or* $\mu_0 \neq x_0^{2^{t/2}}$ then the random combination

$$(x_0^r \cdot \mu_0)^{2^{t/2}} \neq (\mu_0^r \cdot y_0)$$

x_1 y_1

$\hookrightarrow (x_1, y_1, t/2) \notin \mathcal{L}_{f_\square}$

w.h.p. over choice of r. $\square$

Exercise 2

Prove Claims 1 and 2. Argue overall soundness.

Plan for this Session

1. Verifiable Outsourcing of Computation
   - Succinct Non-Interactive Argument (SNARG)

2. SNARG for Repeated Squaring
   - Step I: Interactive Proof for Repeated Squaring
   - Step II: Remove Interaction via Fiat-Shamir Transform

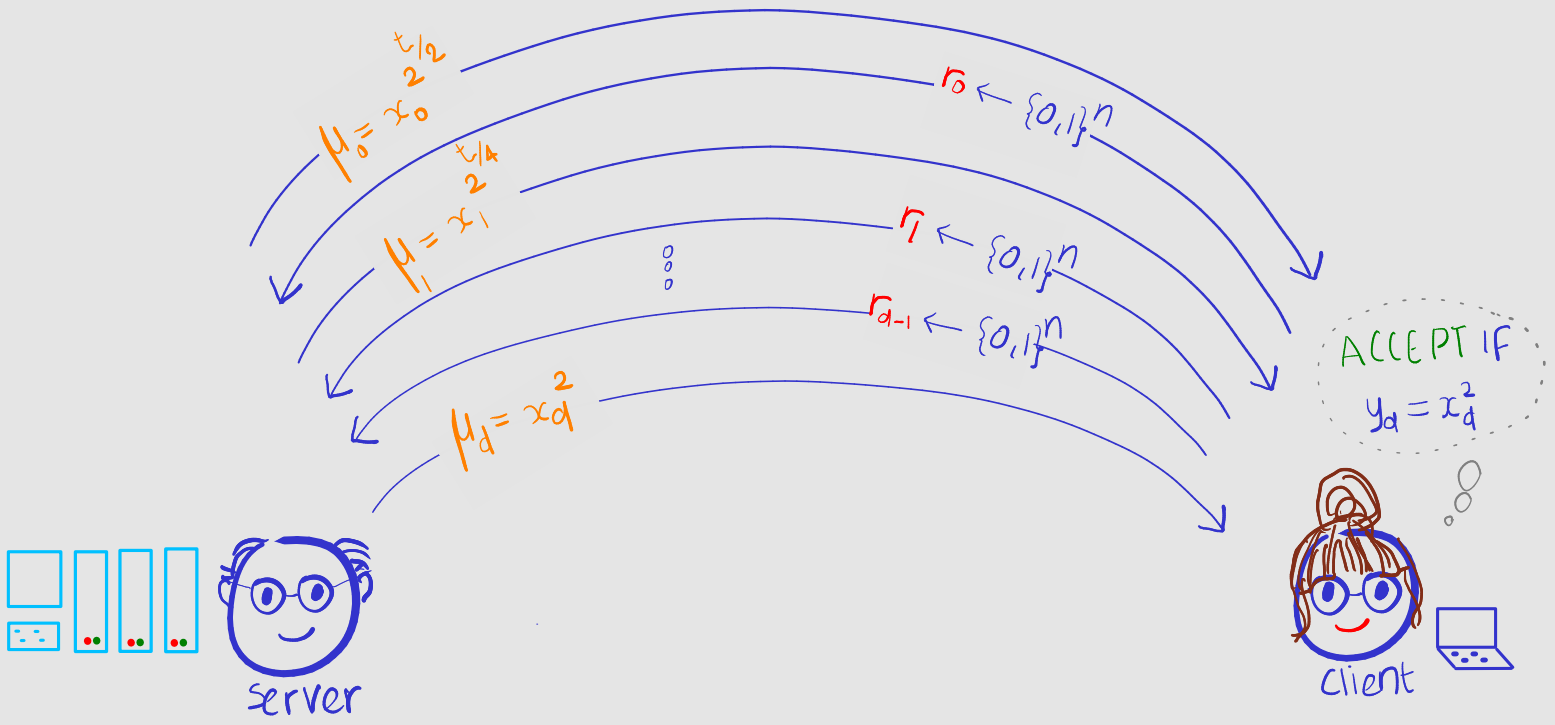
Step II: Apply Fiat-Shamir Transform

- Fiat-Shamir Transform: replace interaction with client (verifier) by calls to random oracle

Step II: Apply Fiat-Shamir Transform

- Fiat-Shamir Transform: replace interaction with client (verifier) by calls to random oracle

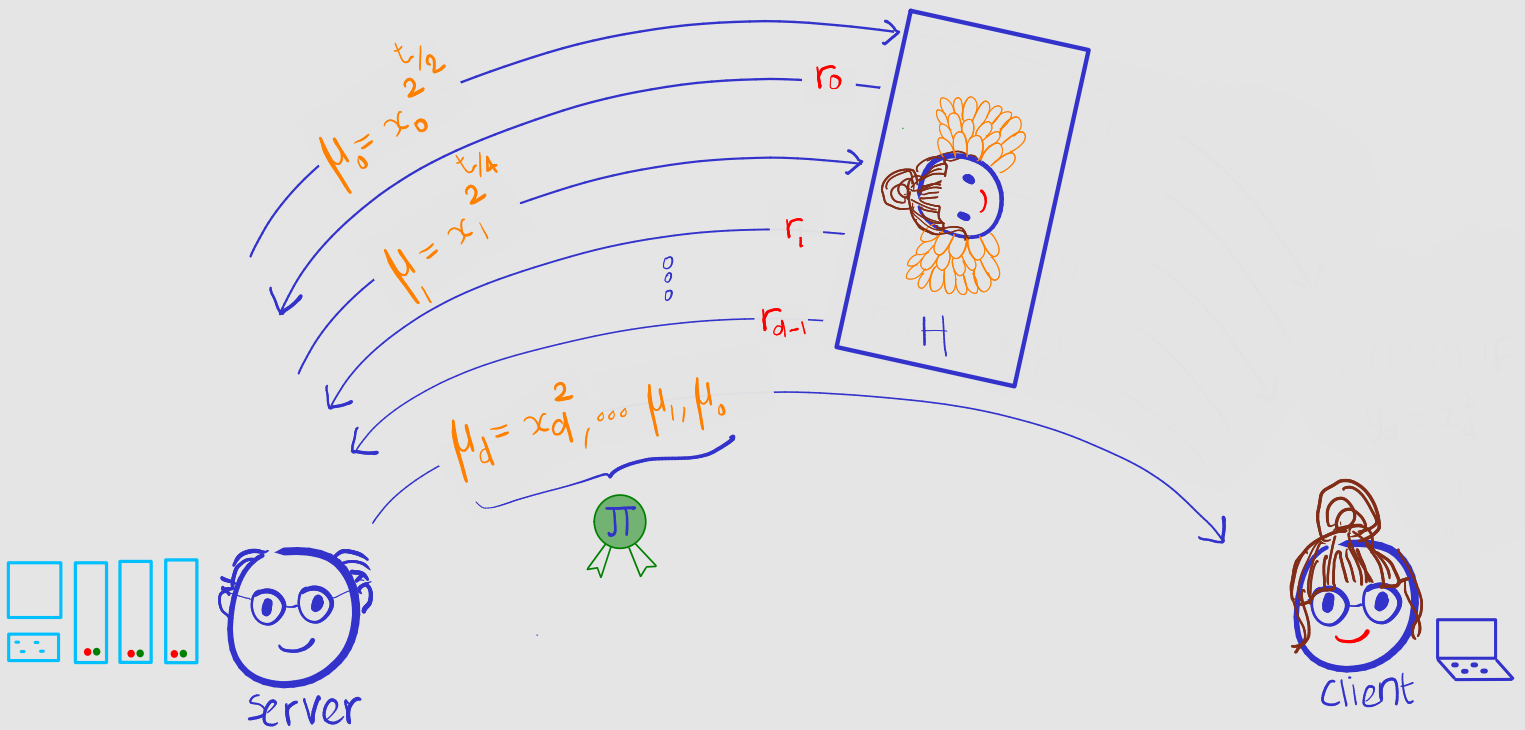
Protocol 2 (Π^H, Π after Fiat-Shamir)



Step II: Apply Fiat-Shamir Transform

- Fiat-Shamir Transform: replace interaction with client (verifier) by calls to random oracle

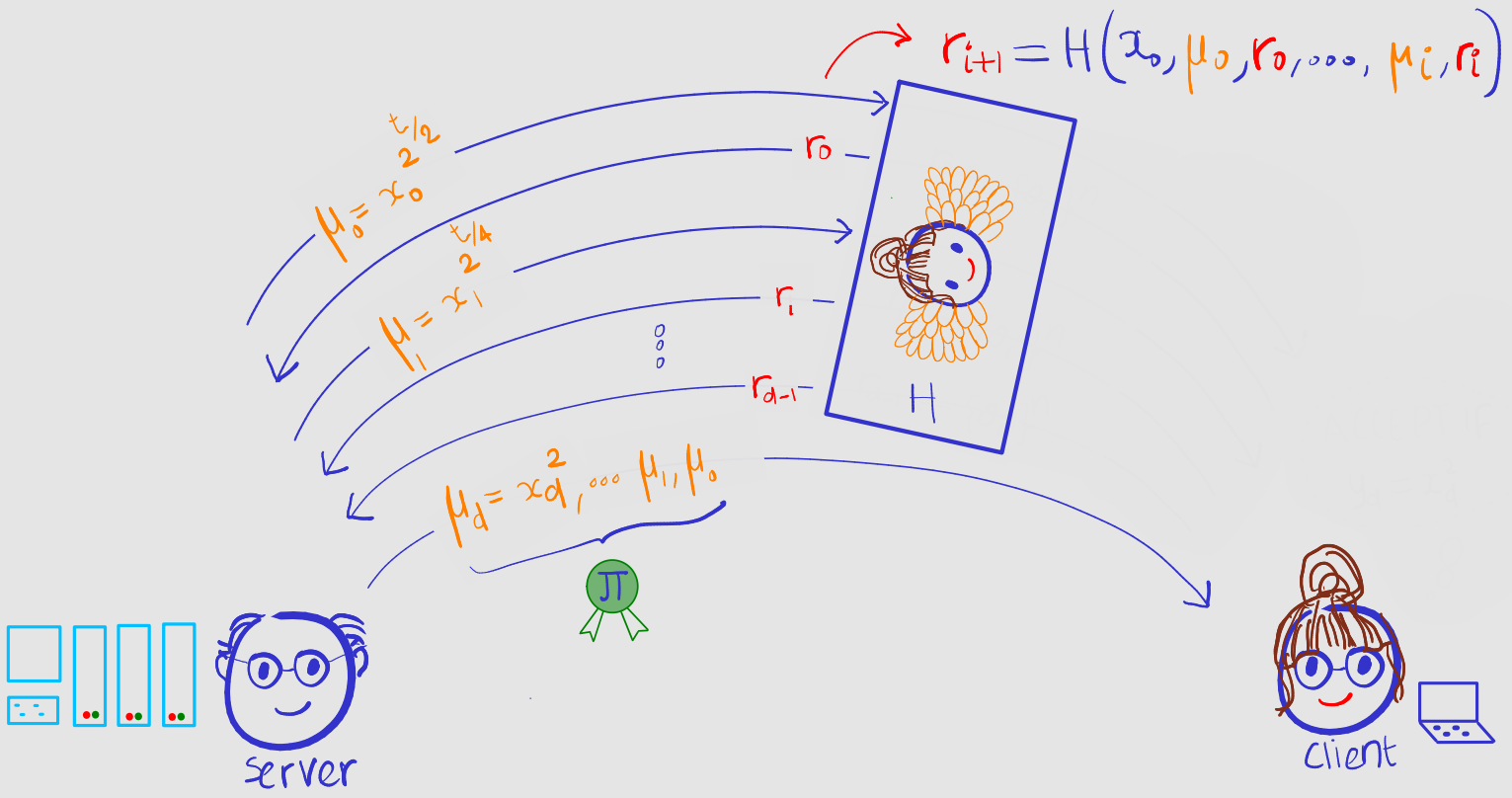
Protocol 2 (Π^H, Π after Fiat-Shamir)



Step II: Apply Fiat-Shamir Transform

- Fiat-Shamir Transform: replace interaction with client (verifier) by calls to random oracle

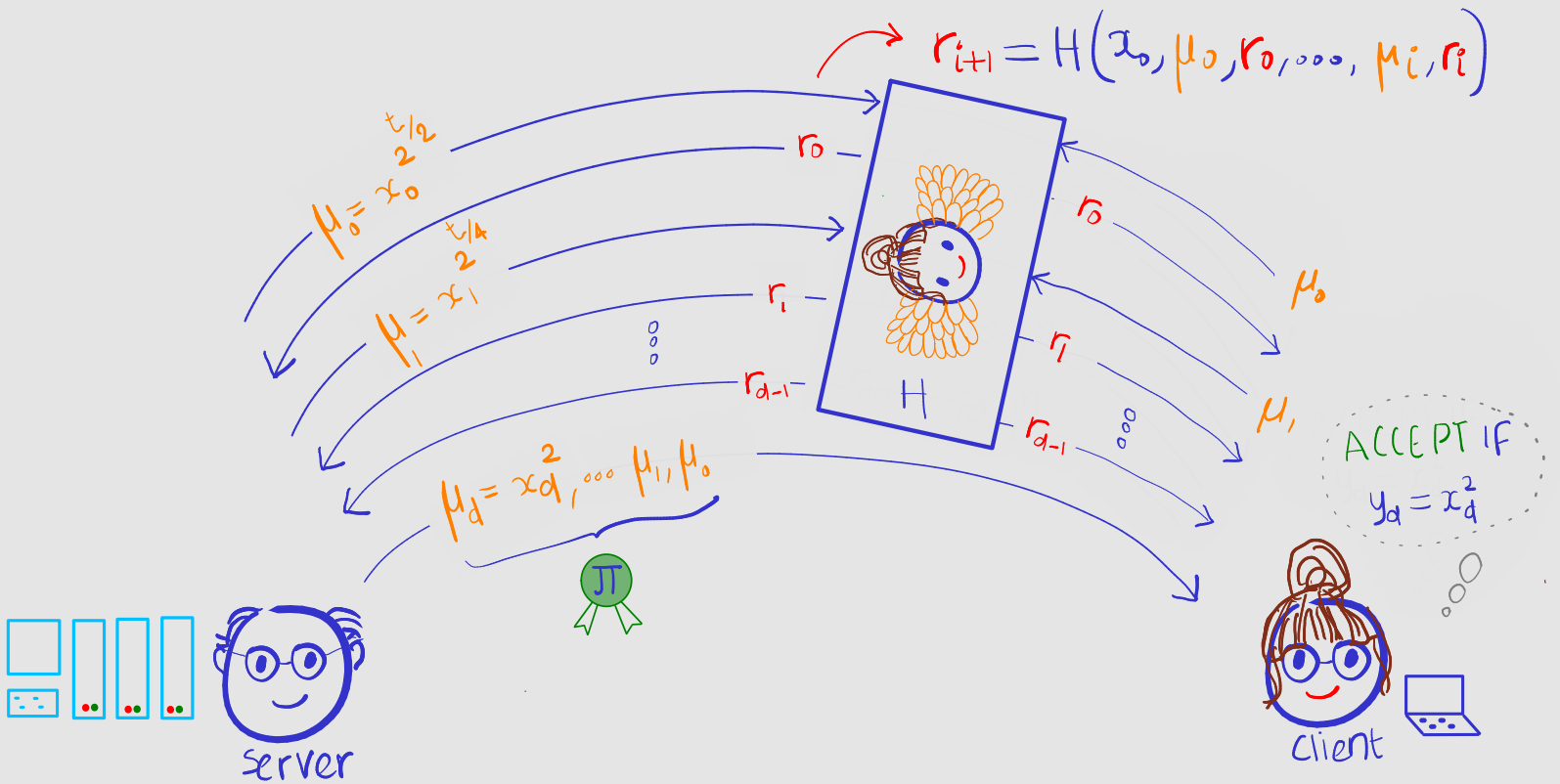
Protocol 2 (Π^H, Π after Fiat-Shamir)



Step II: Apply Fiat-Shamir Transform

- Fiat-Shamir Transform: replace interaction with client (verifier) by calls to random oracle

Protocol 2 (Π^H, Π after Fiat-Shamir)



Why is Π^{H} Sound?

Theorem 2

Π^{H} is a SNARG for $\mathcal{L}_{\mathsf{f}}$ in random-oracle model.

Proof Sketch.

Statistical soundness (for first round) can be argued using:

1. Claim: if $y_0 \neq x_0^{2^t}$ then $y_0 \neq \mu_0^{2^{t/2}}$ *or* $\mu_0 \neq x_0^{2^{t/2}}$ must hold
2. Claim: if $y_0 \neq \mu_0^{2^{t/2}}$ *or* $\mu_0 \neq x_0^{2^{t/2}}$ then the random combination

$$(x_0^r \cdot \mu_0)^{2^{t/2}} \neq (\mu_0^r \cdot y_0)$$

w.h.p. over choice of r. □

Why is Π^{H} Sound?

Theorem 2

Π^{H} is a SNARG for $\mathcal{L}_{\mathsf{f}}$ in random-oracle model.

Proof Sketch.

Statistical soundness (for first round) can be argued using:

✓ 1 Claim: if $y_0 \neq x_0^{2^t}$ then $y_0 \neq \mu_0^{2^{t/2}}$ *or* $\mu_0 \neq x_0^{2^{t/2}}$ must hold

✓ 2 Claim: if $y_0 \neq \mu_0^{2^{t/2}}$ *or* $\mu_0 \neq x_0^{2^{t/2}}$ then the random combination

$$(x_0^r \cdot \mu_0)^{2^{t/2}} \neq (\mu_0^r \cdot y_0)$$

w.h.p. over choice of r. ↖ holds also for RO □

Why is Π^{H} Sound?

Theorem 2

Π^{H} is a SNARG for $\mathcal{L}_{\mathsf{f}}$ in random-oracle model.

Proof Sketch.

Computational ~~Statistical~~ soundness (for first round) can be argued using:

✓ 1 Claim: if $y_0 \neq x_0^{2^t}$ then $y_0 \neq \mu_0^{2^{t/2}}$ *or* $\mu_0 \neq x_0^{2^{t/2}}$ must hold

✓ 2 Claim: if $y_0 \neq \mu_0^{2^{t/2}}$ *or* $\mu_0 \neq x_0^{2^{t/2}}$ then the random combination

$$(x_0^r \cdot \mu_0)^{2^{t/2}} \neq (\mu_0^r \cdot y_0)$$

w.h.p. over choice of r. ↖ holds also for RO

□

Why is Π^{H} Sound?

Theorem 2

Π^{H} is a SNARG for $\mathcal{L}_{\mathsf{f}}$ in random-oracle model.

Proof Sketch.

Computational ~~Statistical~~ soundness (for first round) can be argued using:

✓ 1 Claim: if $y_0 \neq x_0^{2^t}$ then $y_0 \neq \mu_0^{2^{t/2}}$ *or* $\mu_0 \neq x_0^{2^{t/2}}$ must hold

✓ 2 Claim: if $y_0 \neq \mu_0^{2^{t/2}}$ *or* $\mu_0 \neq x_0^{2^{t/2}}$ then the random combination

$$(x_0^r \cdot \mu_0)^{2^{t/2}} \neq (\mu_0^r \cdot y_0)$$

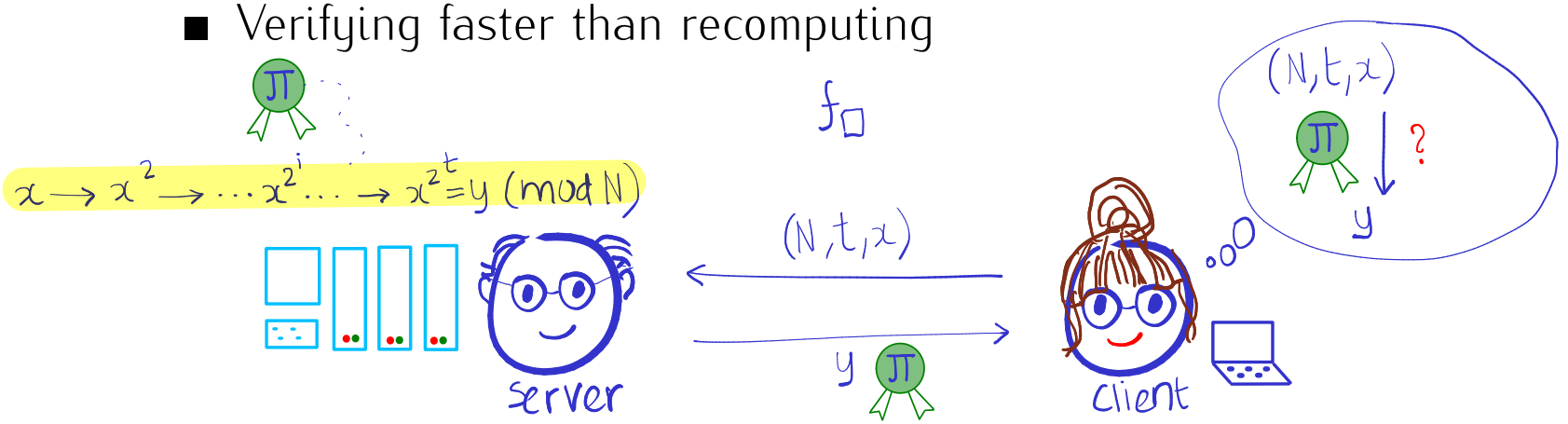
w.h.p. over choice of r. ↖ holds also for RO □

Exercise 3

Work out the proof of soundness in random-oracle model.

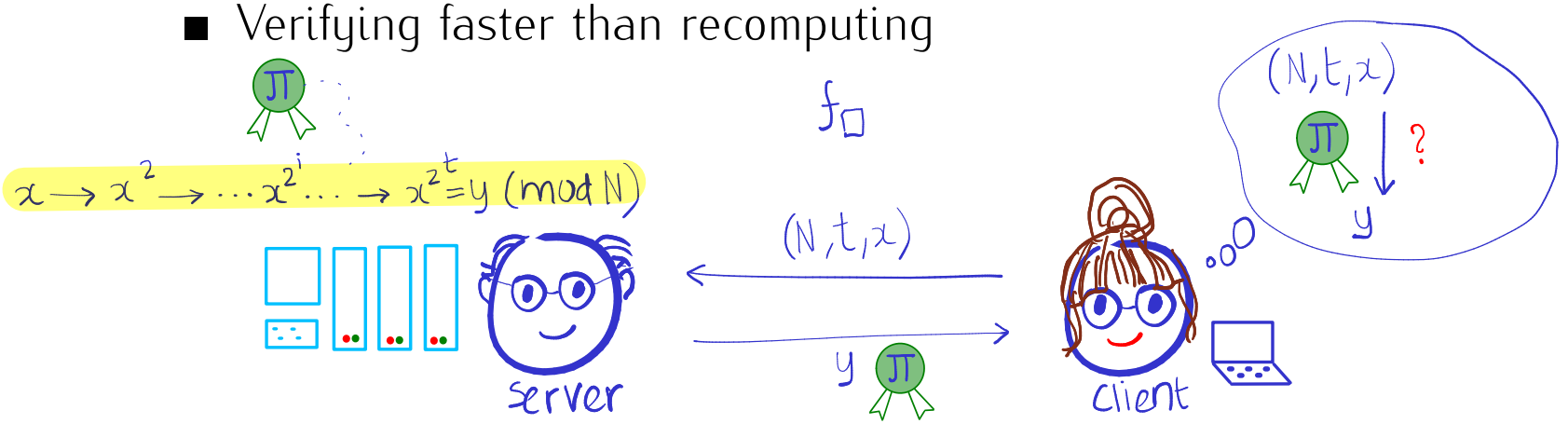
To Recap Today's Lecture

- Task 7: secure outsourcing in the client-server setting
- Task 7.b: verifiable outsourcing in the client-server setting
  - Verifying faster than recomputing



To Recap Today's Lecture

- Task 7: secure outsourcing in the client-server setting
- Task 7.b: verifiable outsourcing in the client-server setting
  - Verifying faster than recomputing

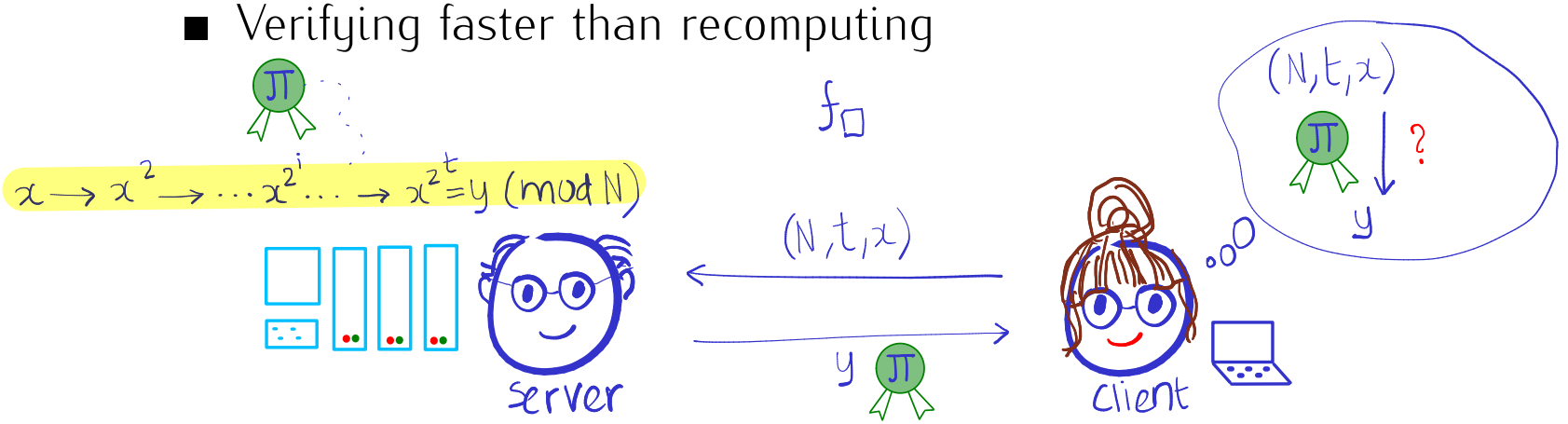


- Key tool: succinct non-interactive argument (SNARG)
- SNARG for repeated squaring problem in RSA group

  - Pietrzak's interactive protocol
  - Main ideas: 1) downward self-reducibility 2) folding
  - SNARG via Fiat-Shamir transform

To Recap Today's Lecture

- Task 7: secure outsourcing in the client-server setting
- Task 7.b: verifiable outsourcing in the client-server setting
  - Verifying faster than recomputing



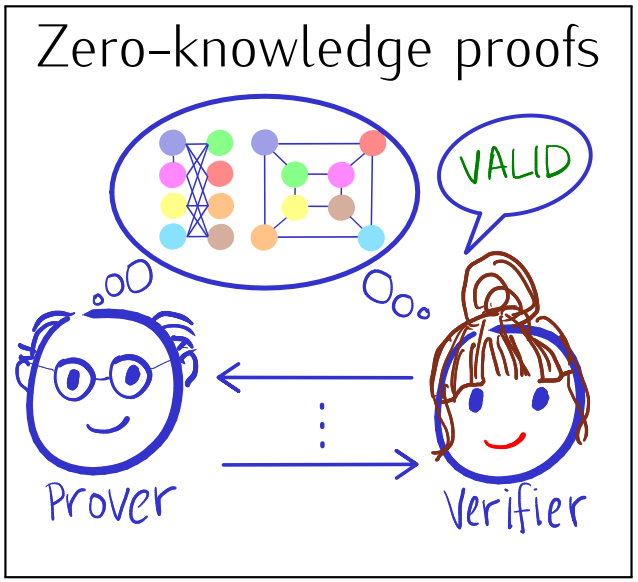
- Key tool: succinct non-interactive argument (SNARG)
- SNARG for repeated squaring problem in RSA group

  - Pietrzak's interactive protocol
  - Main ideas: 1) downward self-reducibility 2) folding
  - SNARG via Fiat-Shamir transform

- SNARGs for **NP** (in ROM)
  - Coming Spring: Introduction to Probabilistic Proof Systems

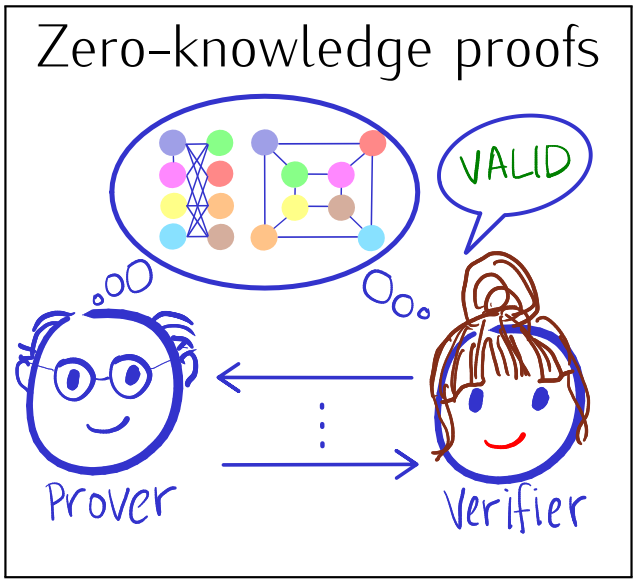
To Recap Module III

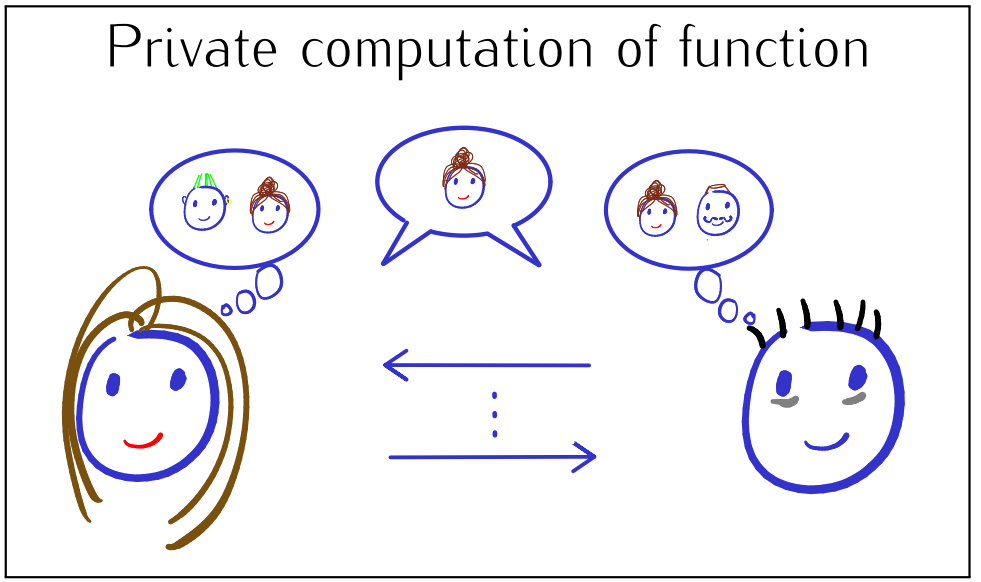
- We saw several avatars of secure computation



To Recap Module III

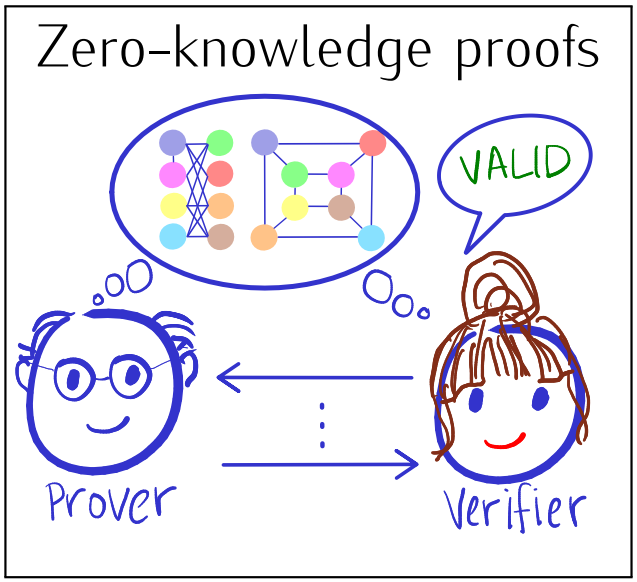
- We saw several avatars of secure computation

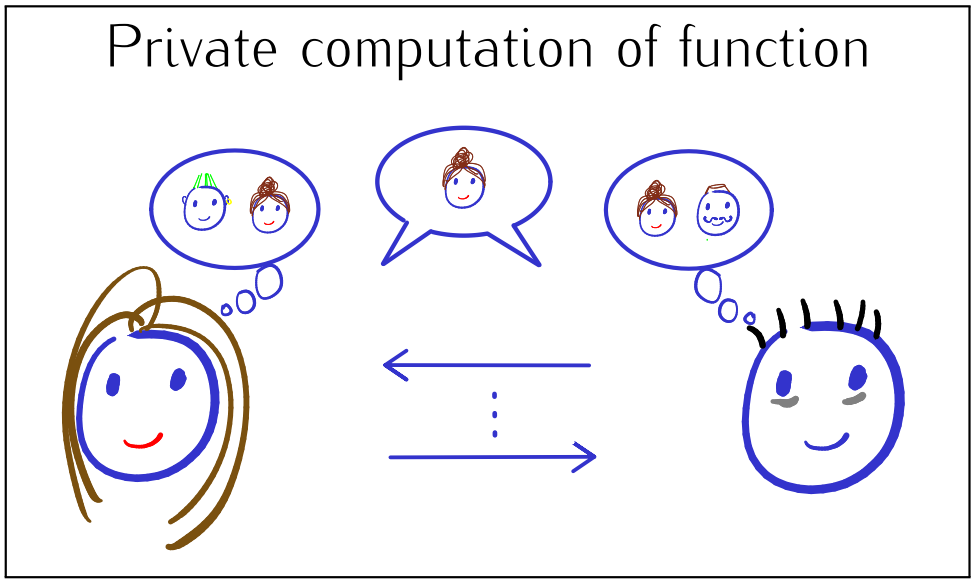


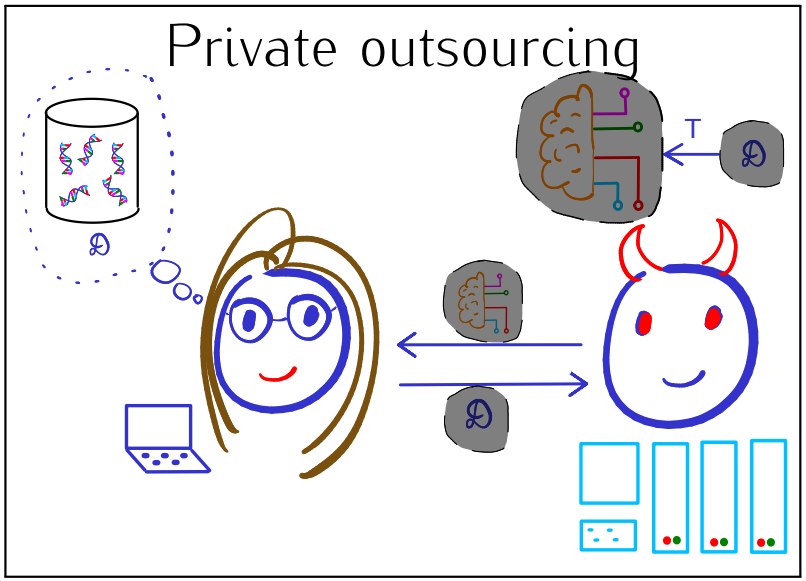


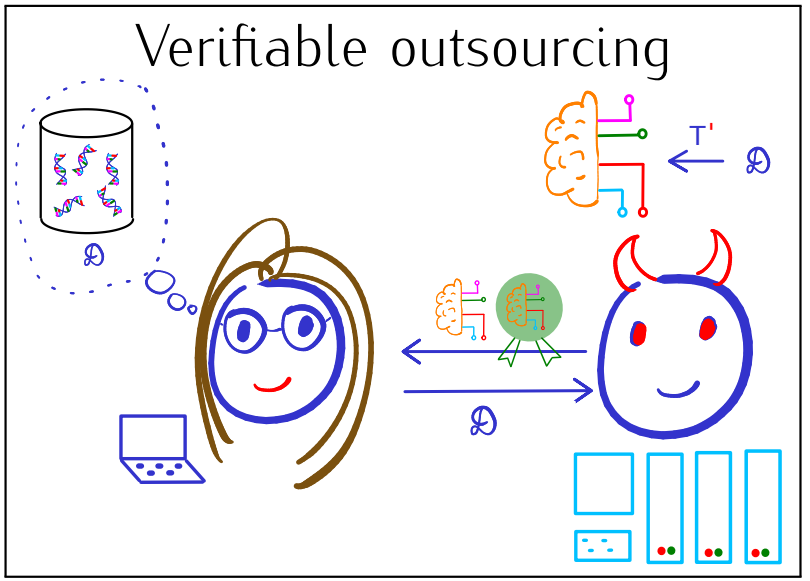
To Recap Module III

- We saw several avatars of secure computation

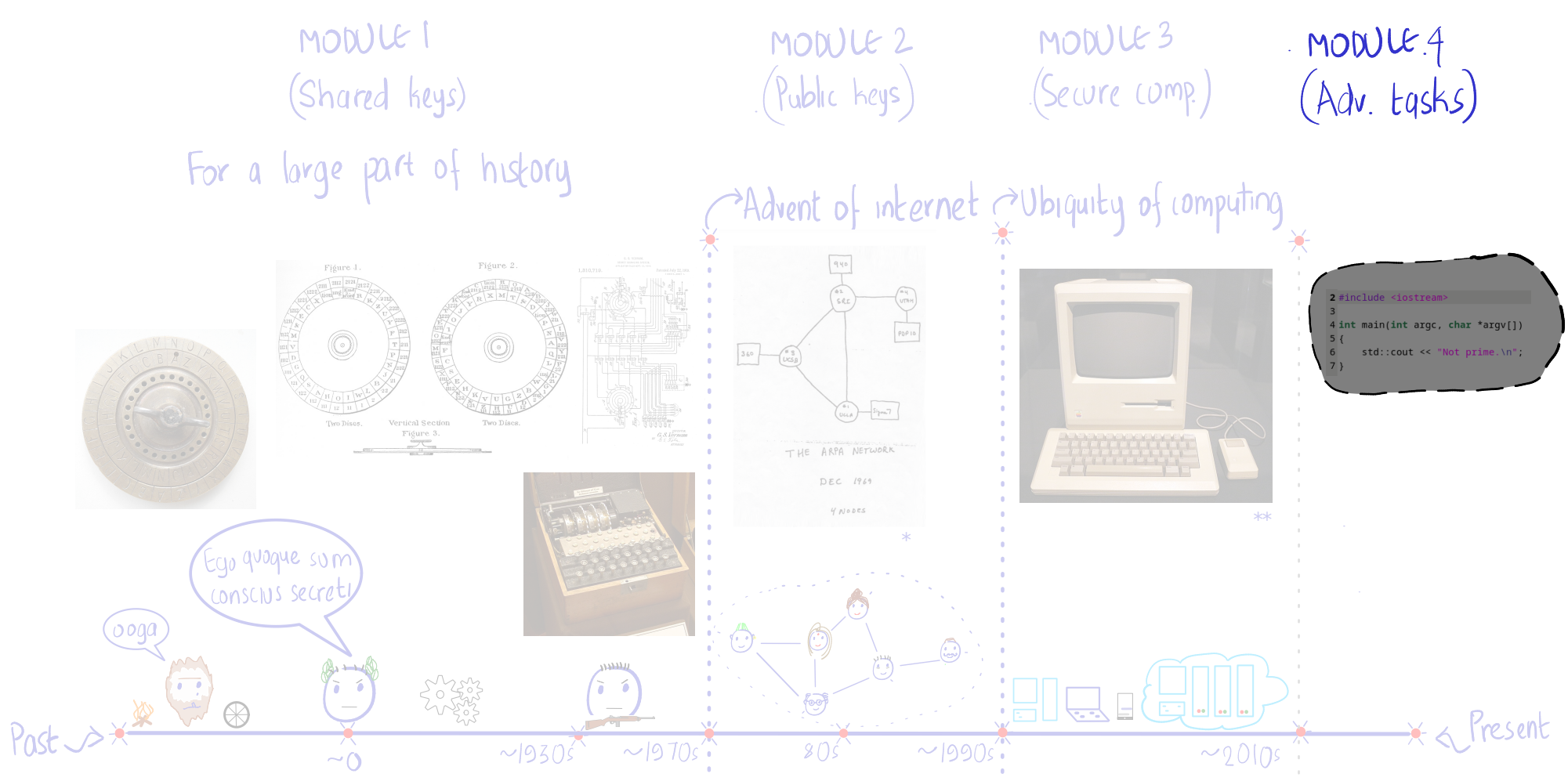








Module IV: Next Few Lectures



- Code obfuscation
  - What cryptography is possible if you can obfuscate code?

Credit for images: Wikipedia (*User:DARPA)(**User:Saliko)

References

1. SNARGs were introduced as "CS proofs" in [Mic94]. You can find a formal definition of SNARG in [BCI+13, §4].
2. The repeated squaring function was introduced in [CLSY93, RSW96] and is extensively used in timed cryptography. For example, the time-lock puzzle from [RSW96] are the verifiable delay functions from [Pie19, Wes19] are both based on *sequential* hardness of repeated squaring.
3. Formal description of Π and Π^{H} can be found in [Pie19]


Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth.

Succinct non-interactive arguments via linear interactive proofs.

In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Heidelberg, March 2013.


J. . Cai, R. J. Lipton, R. Sedgewick, and A. C. . Yao.

Towards uncheatable benchmarks.

In *Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 2–11, May 1993.


Silvio Micali.

CS proofs (extended abstracts).

In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994.


Krzysztof Pietrzak.

Simple verifiable delay functions.

In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019.


R. L. Rivest, A. Shamir, and D. A. Wagner.

Time-lock puzzles and timed-release crypto.

Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1996.

Benjamin Wesolowski.

Efficient verifiable delay functions.

In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg, May 2019.