Subvector Commitments with Optimal Opening Complexity

Estonian-Latvian CS Theory Days 2026

Matteo Campanelli
Offchain Labs & University of Tartu
binarywhales.com

Link to paper

Prelude: “Traditional” Cryptographic Hashing

Prelude: “Traditional” Cryptographic Hashing

H(01010101100…011)

Prelude: “Traditional” Cryptographic Hashing

H(01010101100…011) ⟶ 

Prelude: “Traditional” Cryptographic Hashing

H(01010101100…011) ⟶ 

Security: “different strings will have different fingerprints”

Prelude: “Traditional” Cryptographic Hashing

H(01010101100…011) ⟶ 

Security: “different strings will have different fingerprints”

H(11010101100…011)

Prelude: “Traditional” Cryptographic Hashing

H(01010101100…011) ⟶ 

Security: “different strings will have different fingerprints”

H(11010101100…011) H(01010101100…001)

Prelude: “Traditional” Cryptographic Hashing

H(01010101100…011) ⟶ 

Security: “different strings will have different fingerprints”

H(11010101100…011) H(01010101100…001) H(01011101100…011)

Prelude: “Traditional” Cryptographic Hashing

H(01010101100…011) 

Security: “different strings will have different fingerprints”

H(11010101100…011)

H(01010101100…001)

H(01011101100…011)

Prelude: “Traditional” Cryptographic Hashing

H(01010101100…011) ⟶ 

Security: “different strings will have different fingerprints”

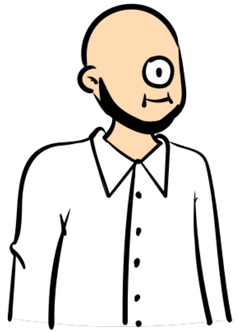
H(11010101100…011) ↓ 

H(01010101100…001) ↓ 

H(01011101100…011) ↓ 

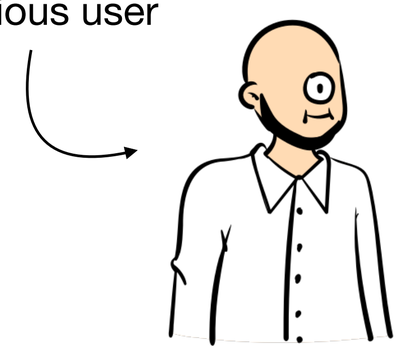
Hashing has a ton of applications to *integrity* in computer systems and in cryptography in general

Limitations of Traditional Hashing



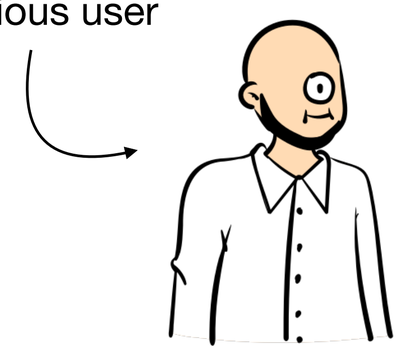
Limitations of Traditional Hashing

A curious user



Limitations of Traditional Hashing

A curious user



Cryptographic hash function

31 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.

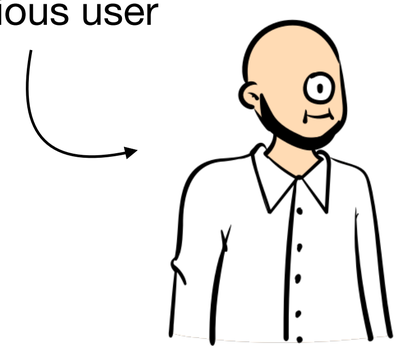
Find sources: "Cryptographic hash function" – news · newspapers · books · scholar · JSTOR *(May 2016)*

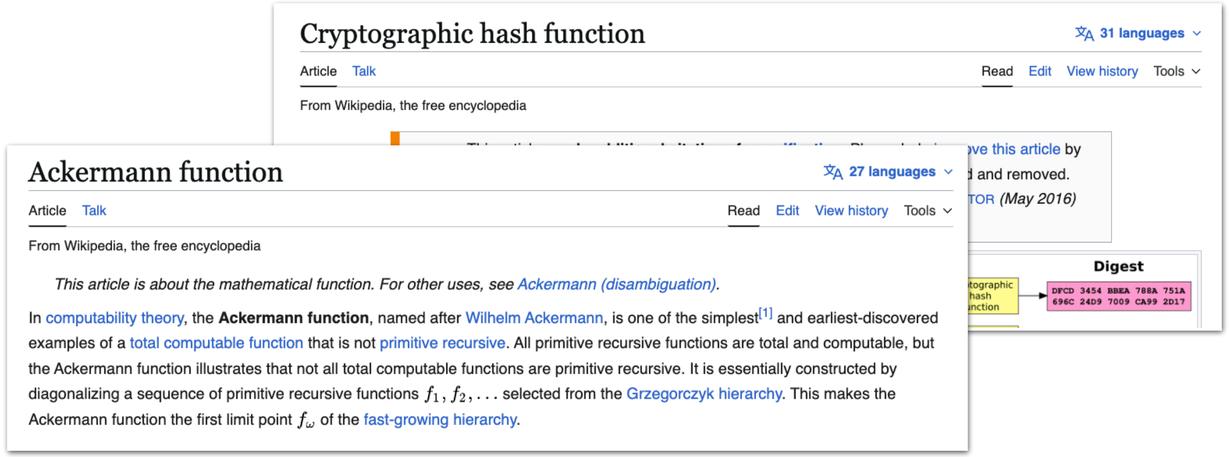
(Learn how and when to remove this message)

Hashing is a one-directional mathematical operation which is quick to calculate, yet hard to reverse.[1] So password storage and digital signtures benefit from hashes.[2] Even a small change in the input results in a very

Limitations of Traditional Hashing

A curious user





Cryptographic hash function

31 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

ve this article by
d and removed.
TOR *(May 2016)*

Digest

DFCD 3454 BBEA 788A 751A
696C 24D9 7009 CA99 2D17

Ackermann function

27 languages

Article Talk Read Edit View history Tools

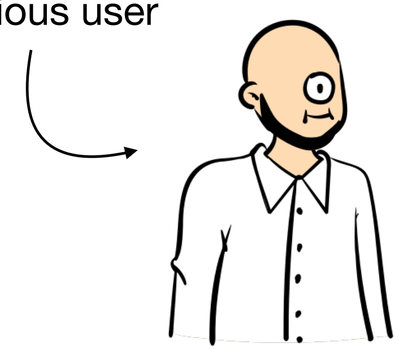
From Wikipedia, the free encyclopedia

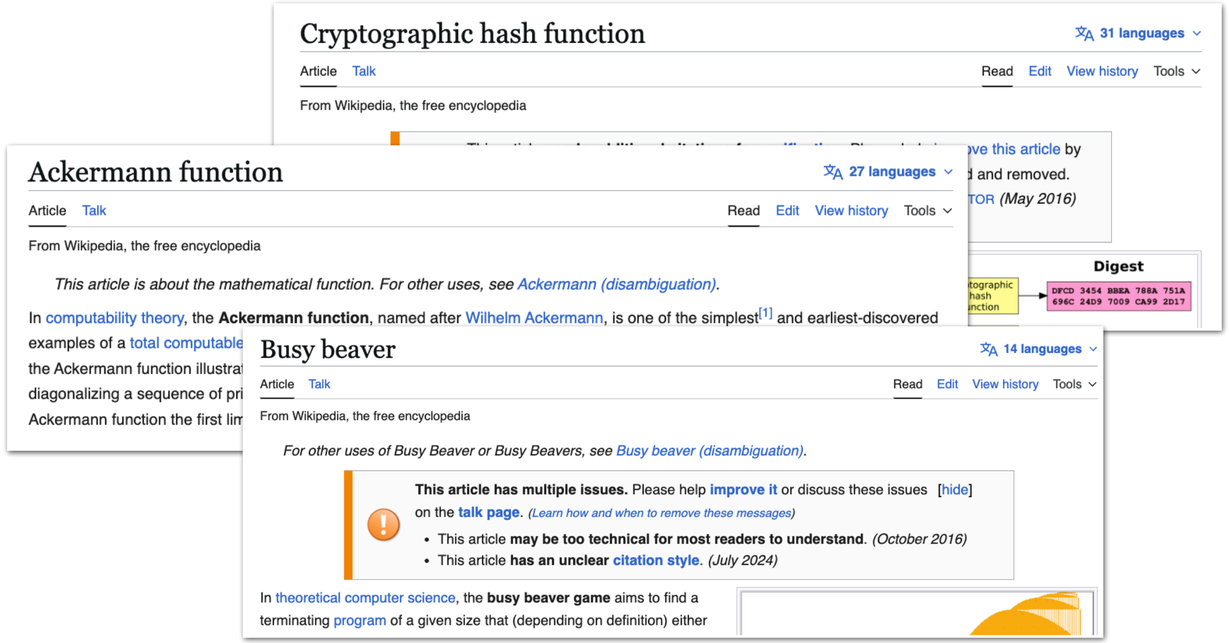
This article is about the mathematical function. For other uses, see Ackermann (disambiguation).

In computability theory, the **Ackermann function**, named after Wilhelm Ackermann, is one of the simplest[1] and earliest-discovered examples of a total computable function that is not primitive recursive. All primitive recursive functions are total and computable, but the Ackermann function illustrates that not all total computable functions are primitive recursive. It is essentially constructed by diagonalizing a sequence of primitive recursive functions $f_1, f_2, \ldots$ selected from the Grzegorczyk hierarchy. This makes the Ackermann function the first limit point f_ω of the fast-growing hierarchy.

Limitations of Traditional Hashing

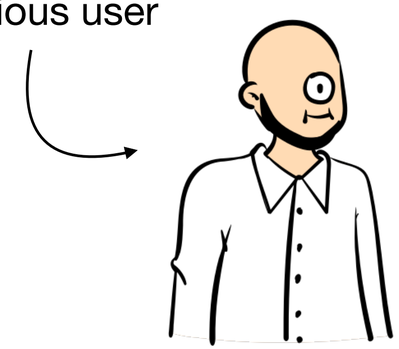
A curious user

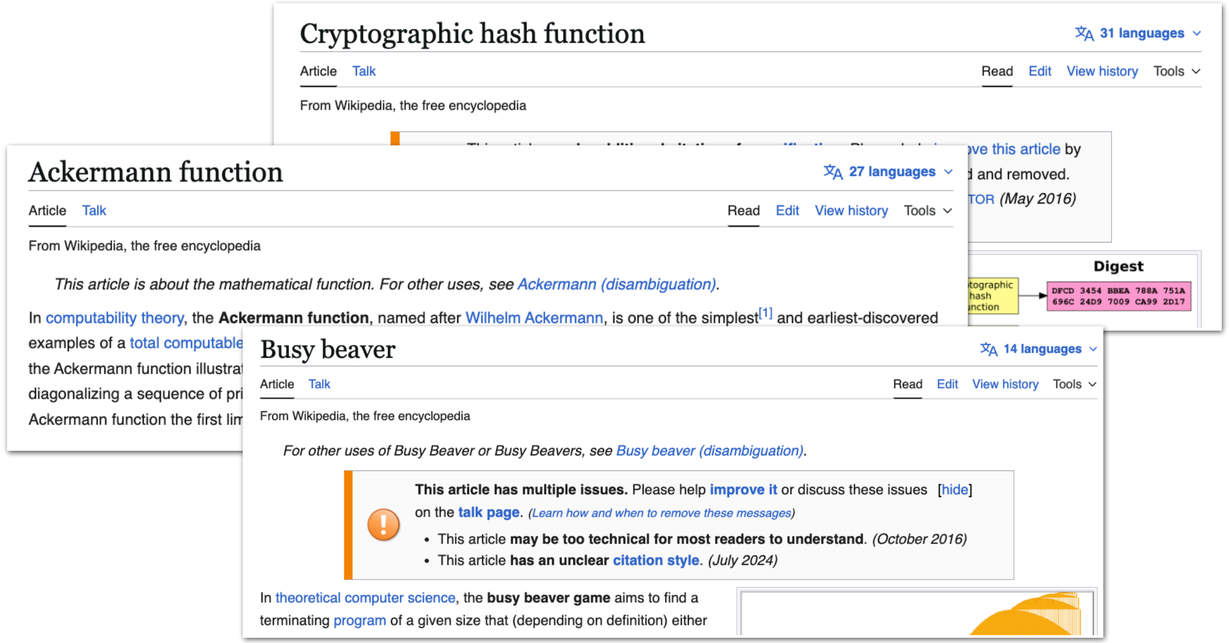


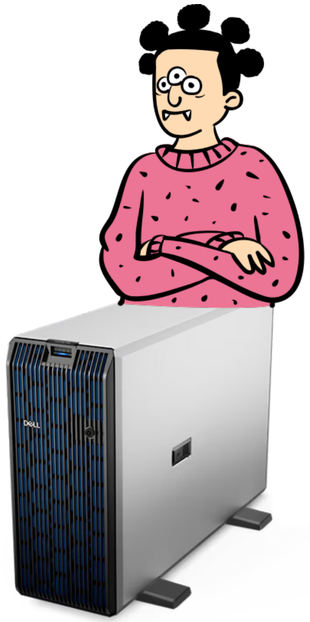


Limitations of Traditional Hashing

A curious user

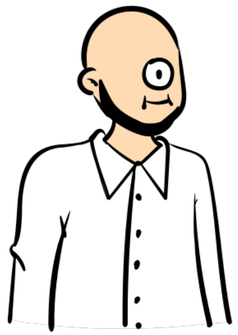


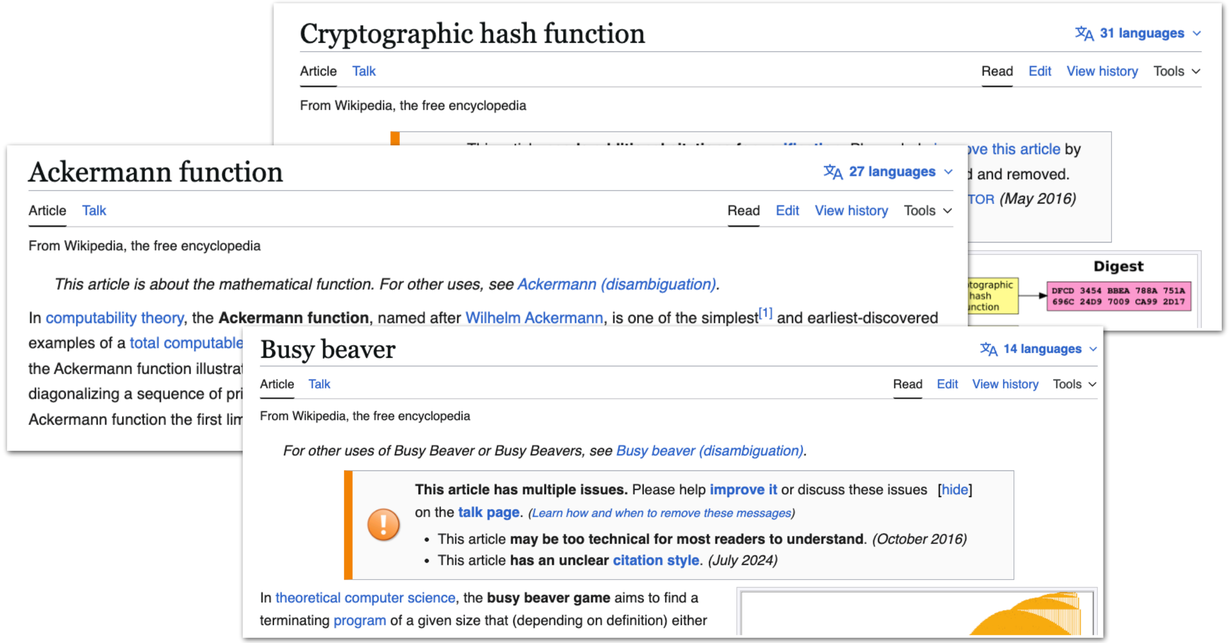


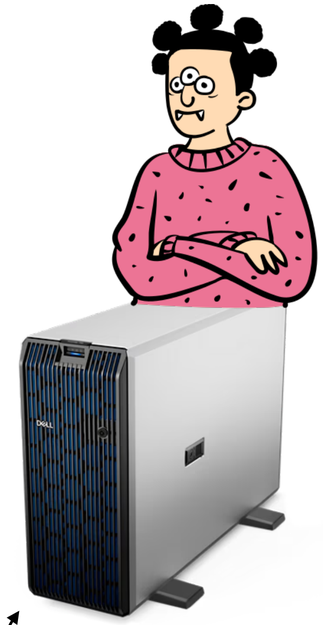


Limitations of Traditional Hashing

A curious user



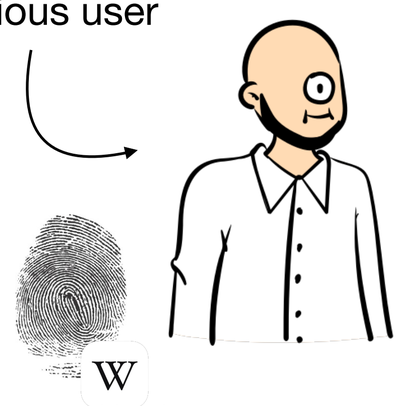




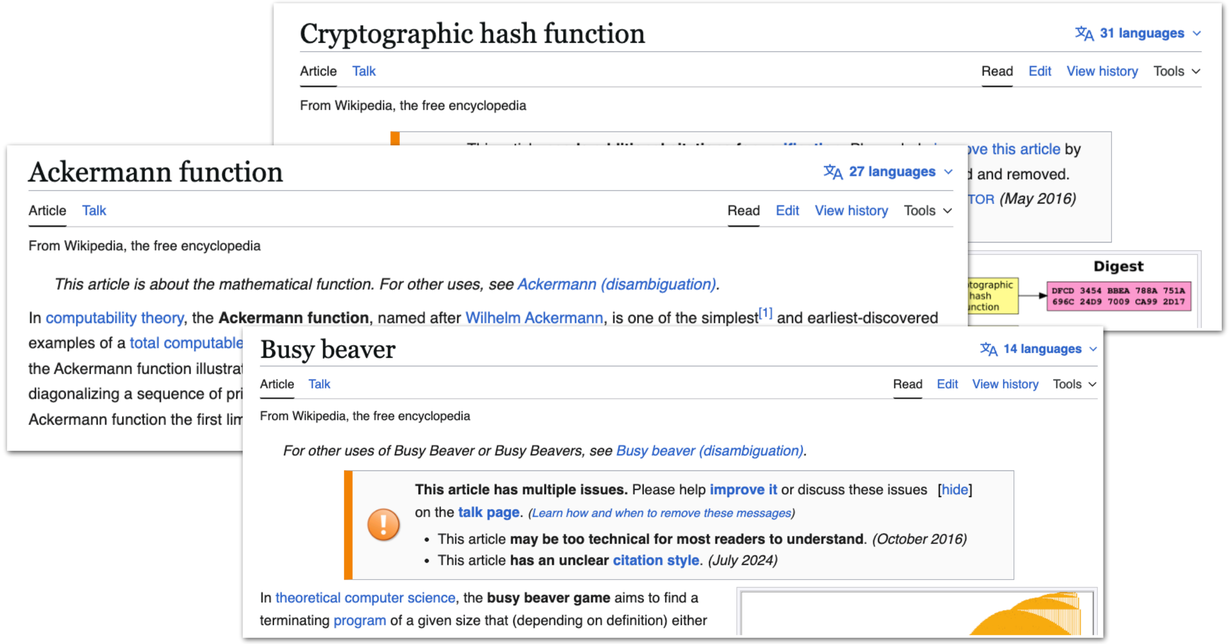
An untrusted server

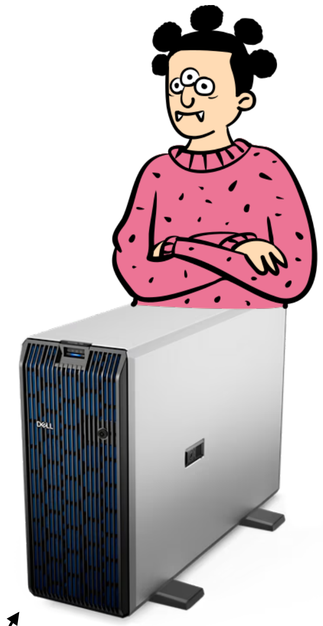
Limitations of Traditional Hashing

A curious user



A trusted hash
of the whole
Wikipedia corpus

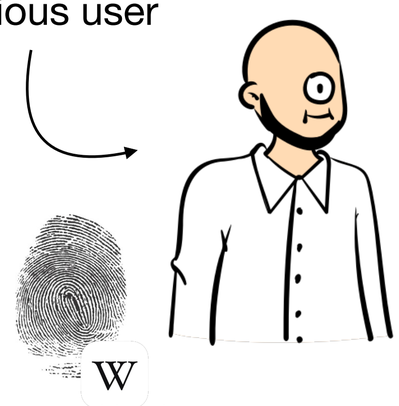




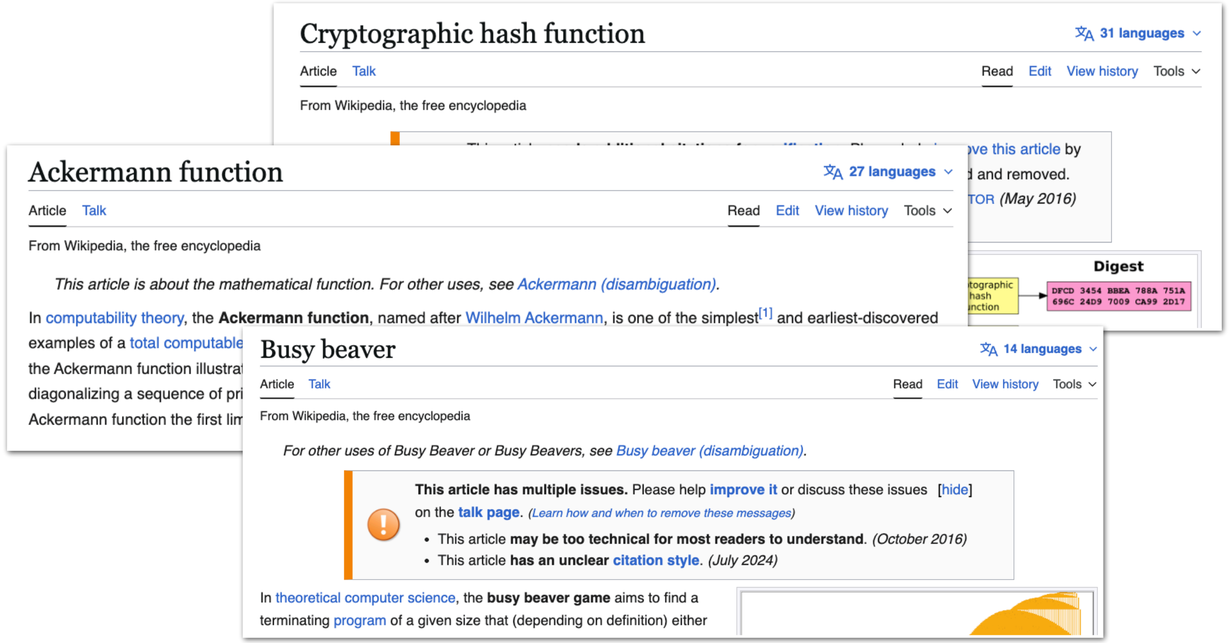
An untrusted server

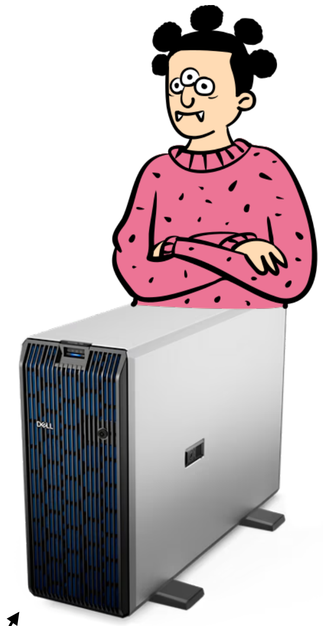
Limitations of Traditional Hashing

A curious user



A trusted hash
of the whole
Wikipedia corpus



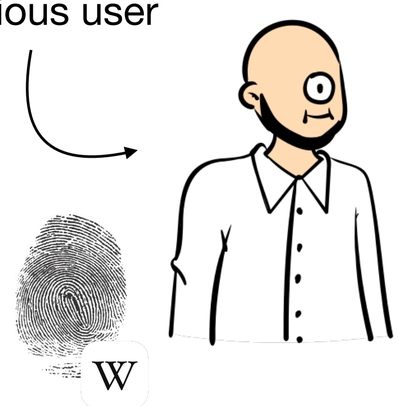


An untrusted server

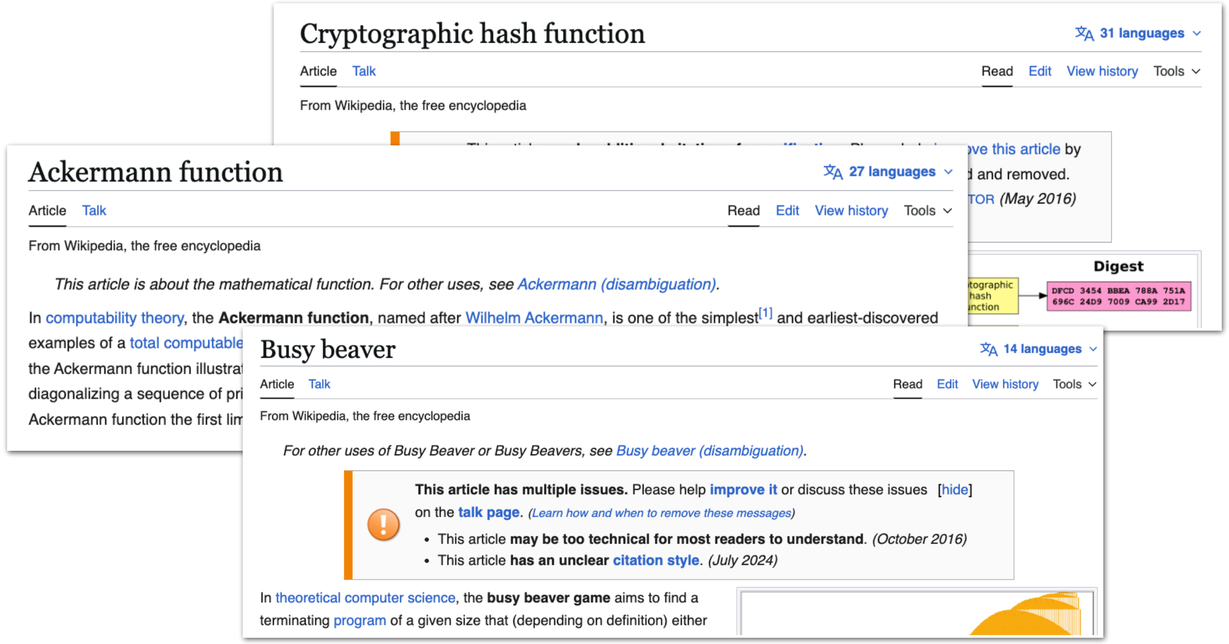
Can we use hashing to check those pages actually come from the Wikipedia corpus?

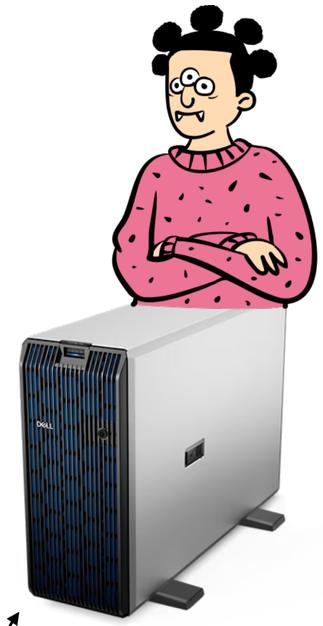
Limitations of Traditional Hashing

A curious user



A trusted hash
of the whole
Wikipedia corpus





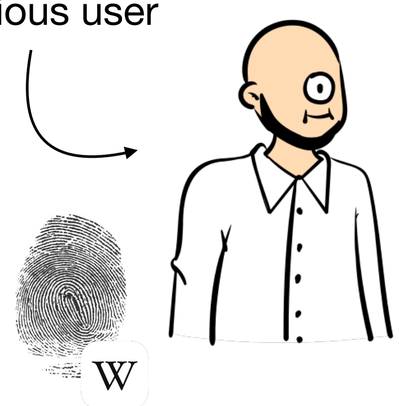
An untrusted server

H( ) ? ==

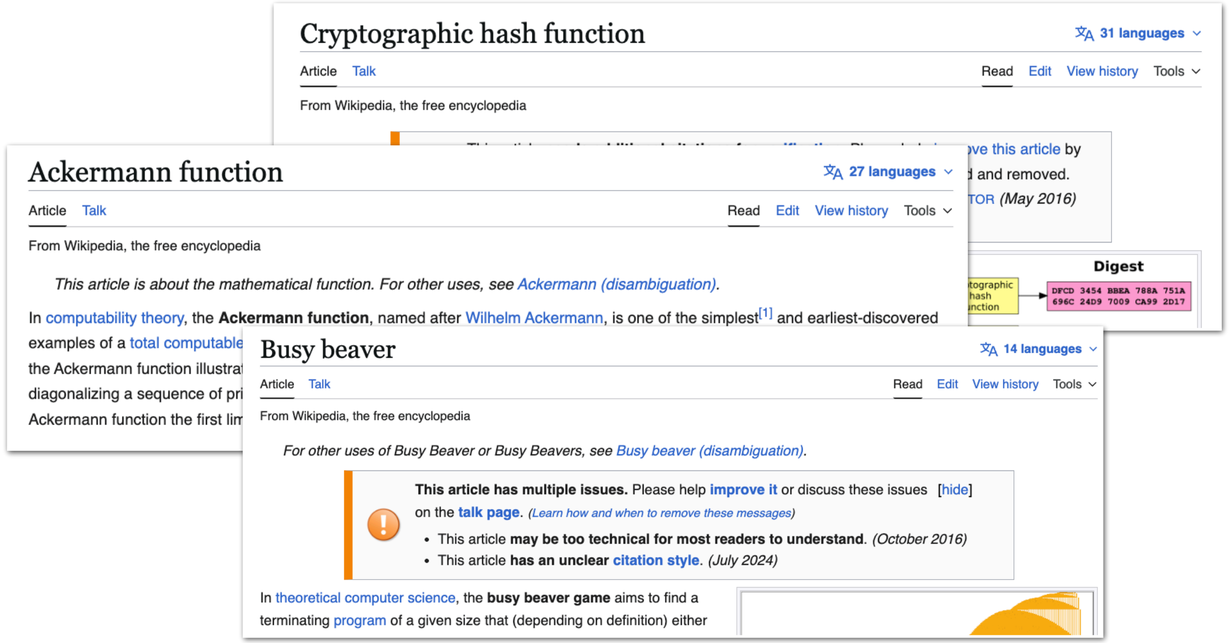
Can we use hashing to check those pages actually come from the Wikipedia corpus?

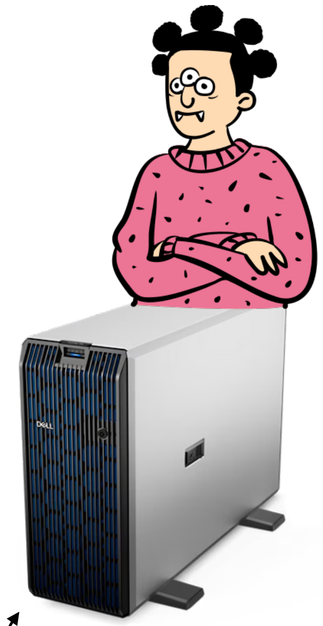
Limitations of Traditional Hashing

A curious user



A trusted hash
of the whole
Wikipedia corpus





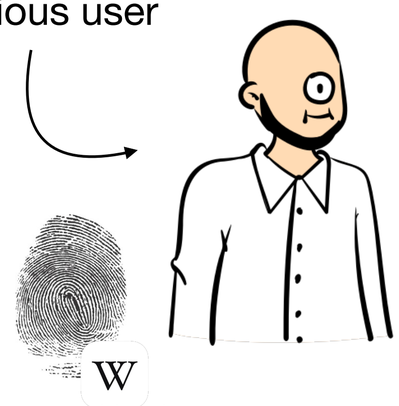
An untrusted server

H() ?== 

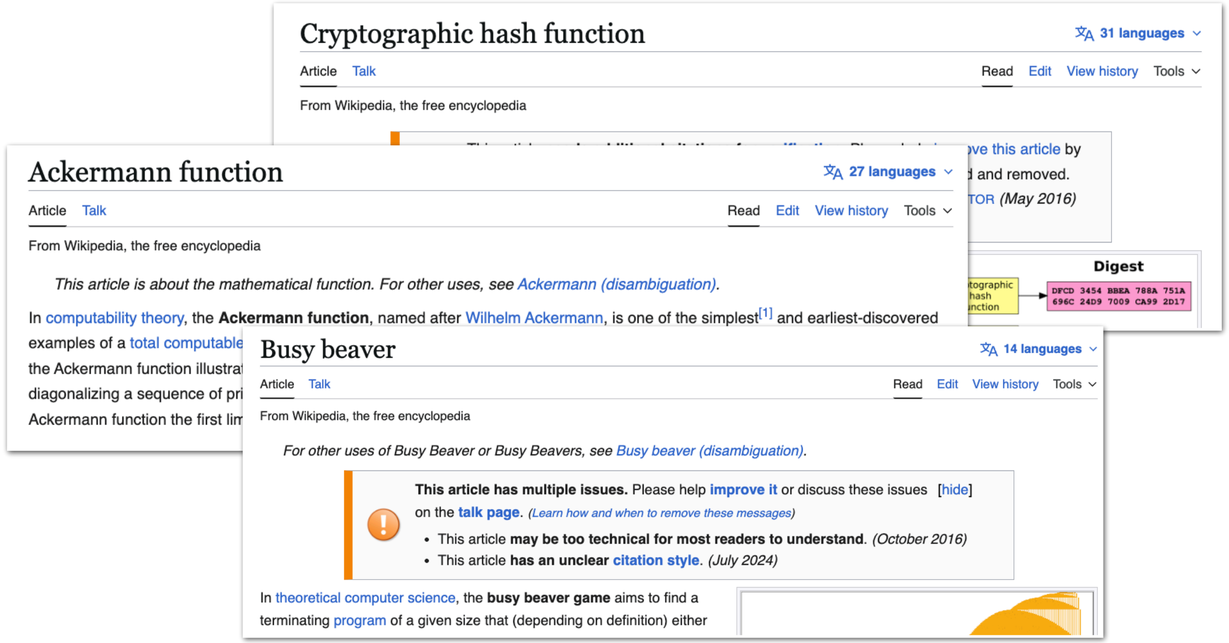
Can we use hashing to check those pages actually come from the Wikipedia corpus?

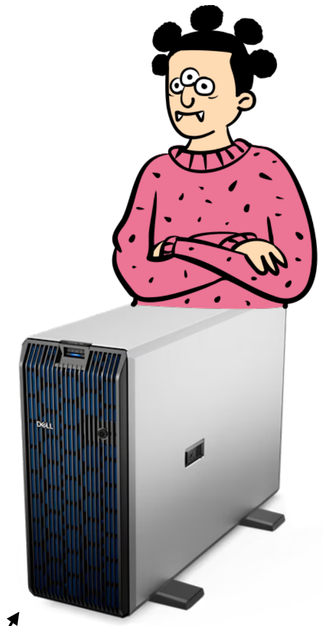
Limitations of Traditional Hashing

A curious user



A trusted hash
of the whole
Wikipedia corpus





An untrusted server

H() ?== 

Not the solution we were looking for

Can we use hashing to check those pages actually come from the Wikipedia corpus?

Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing

Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing

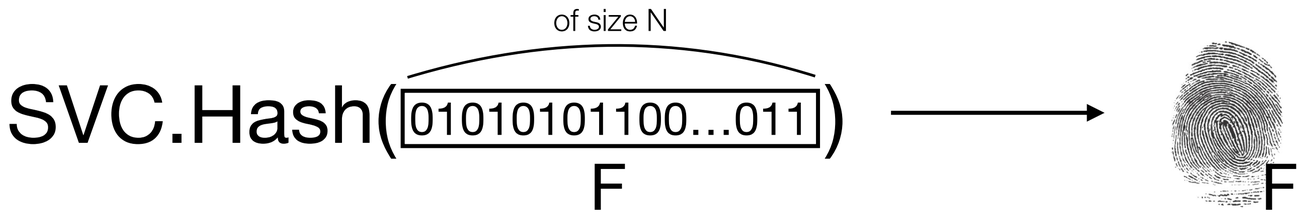
SVC.Hash(01010101100…011) → 

F

F

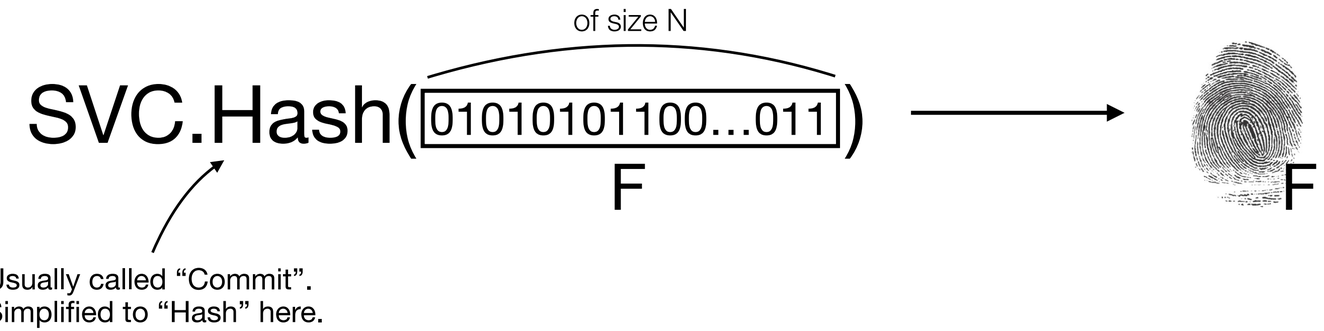
Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing



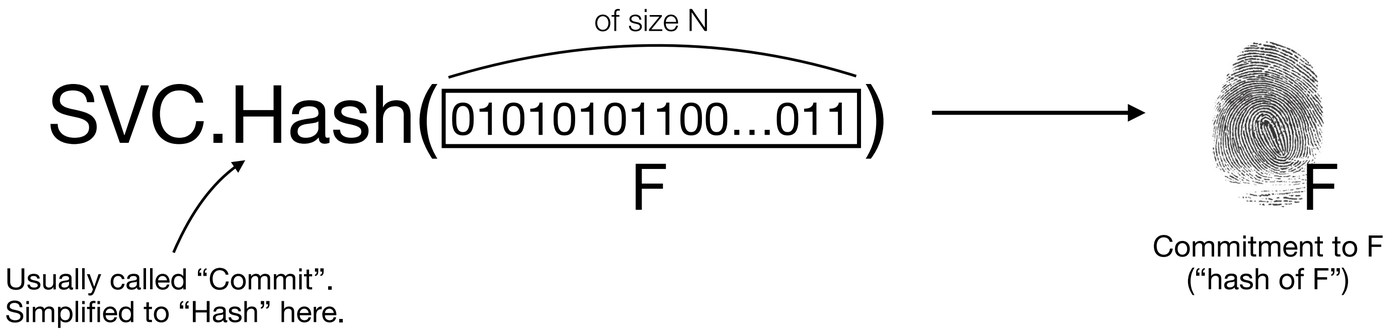
Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing



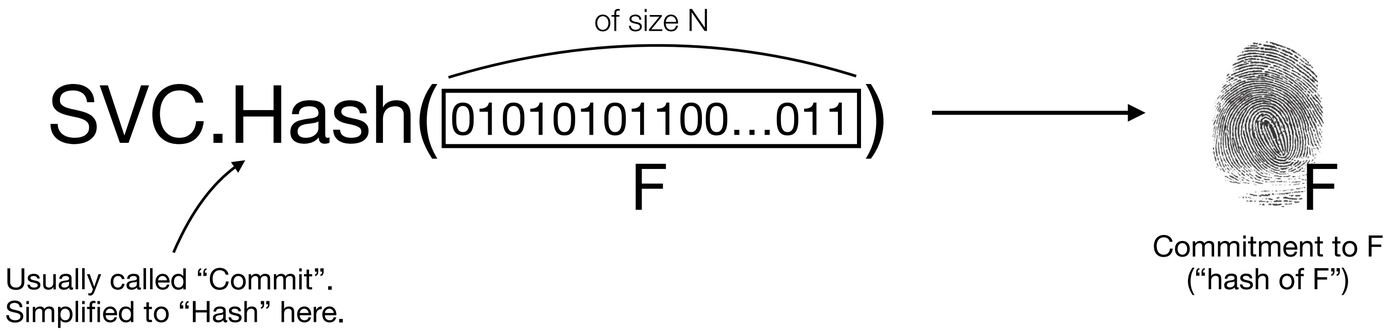
Subvector Commitments (SVC) [LY10,CF13,LM19]

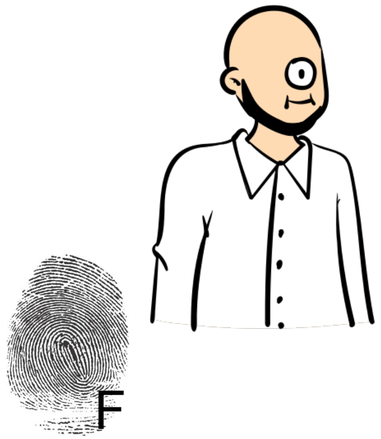
A Fine-Grained Form of Hashing



Subvector Commitments (SVC) [LY10,CF13,LM19]

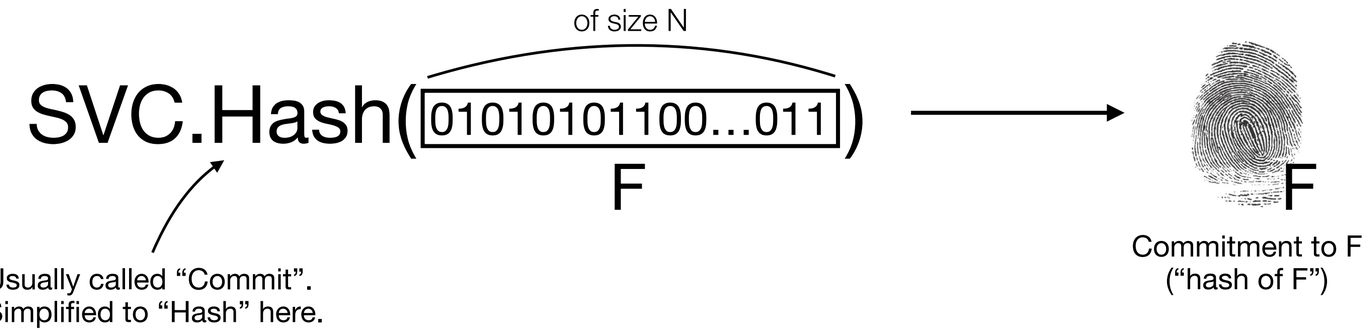
A Fine-Grained Form of Hashing

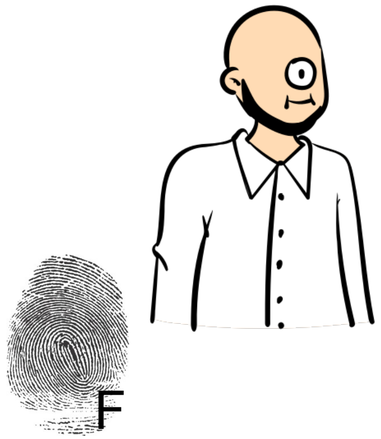


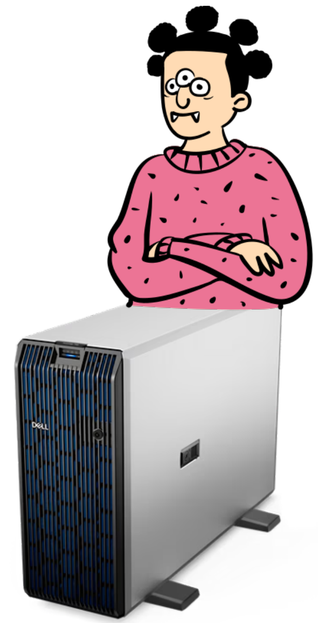


Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing

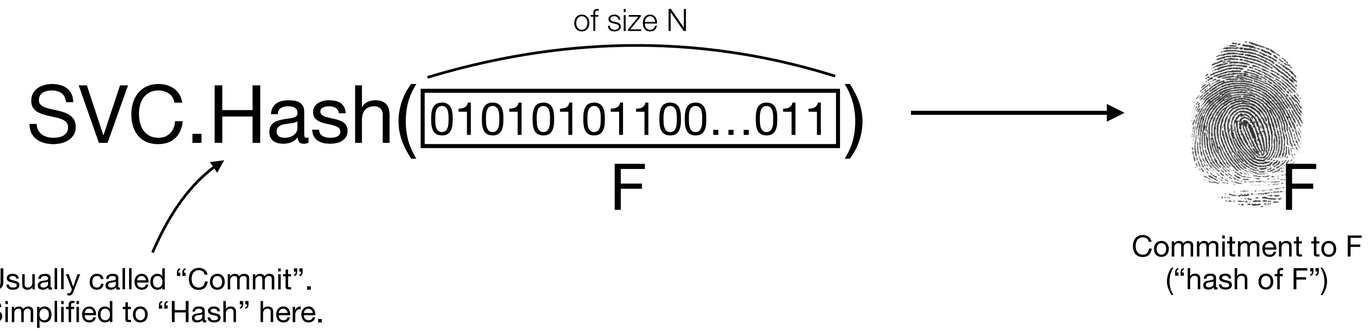


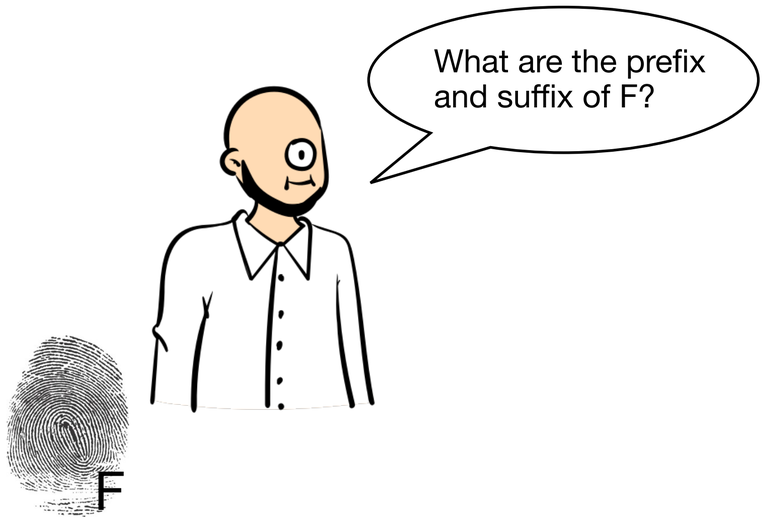


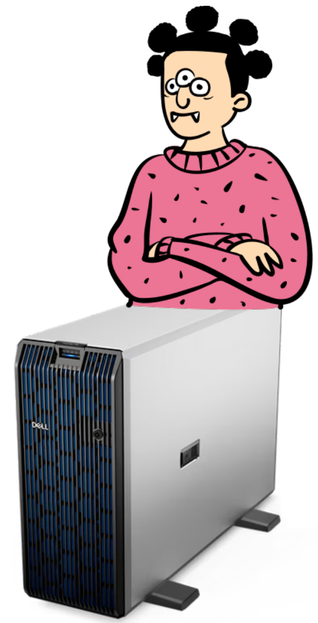


Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing

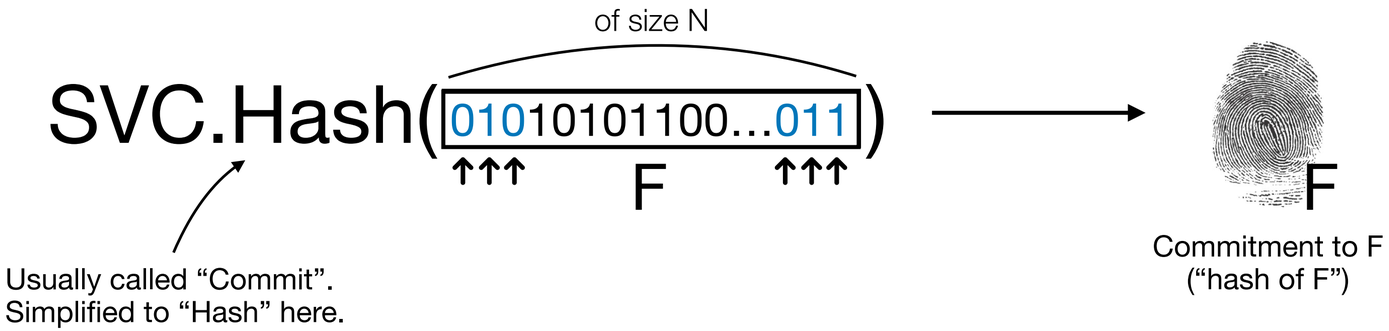


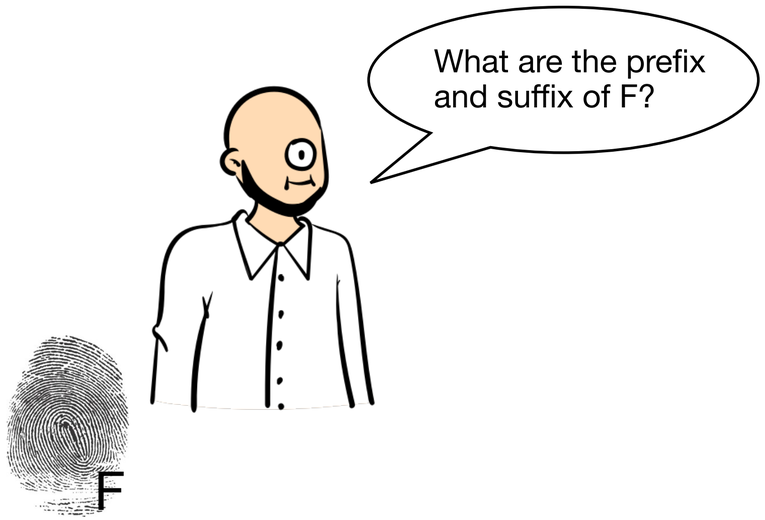


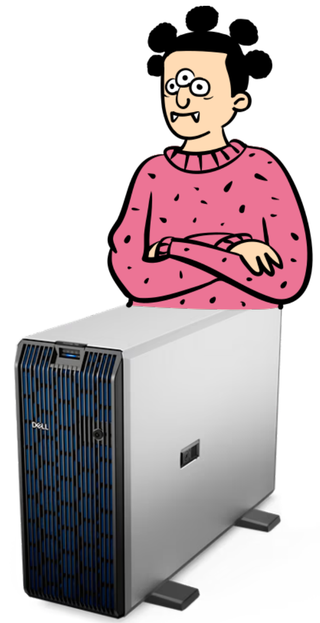


Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing

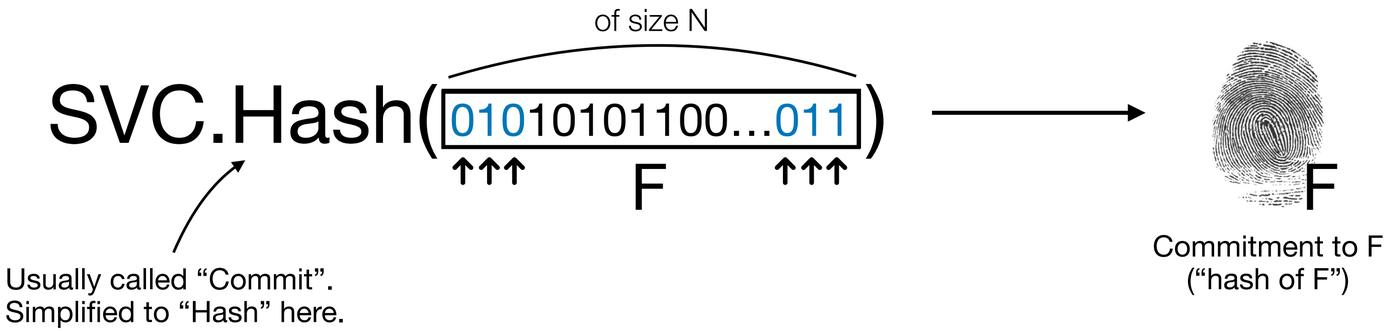


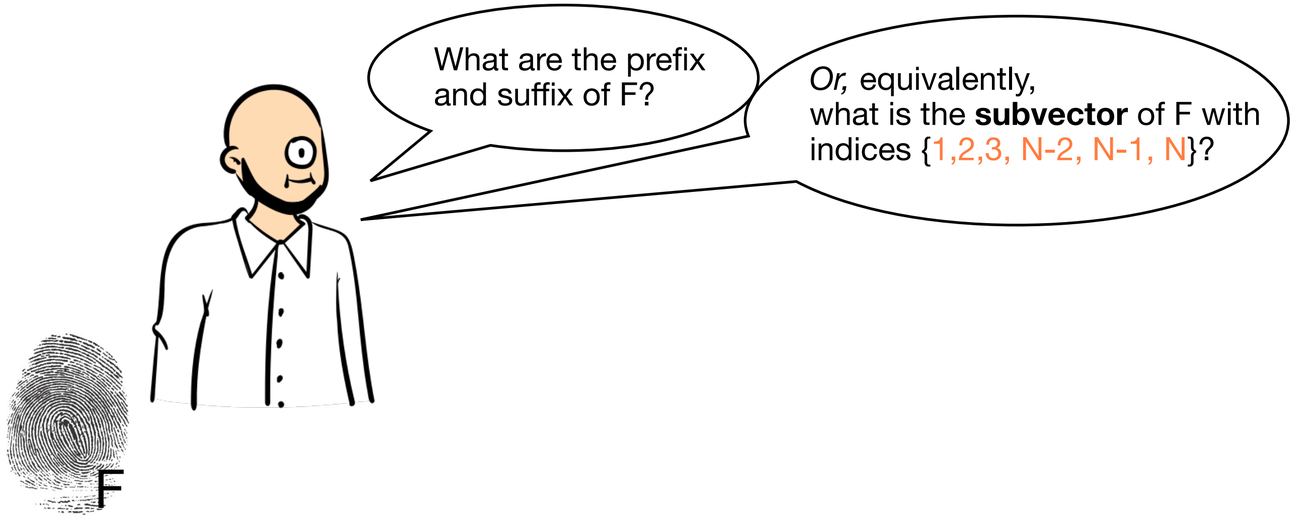


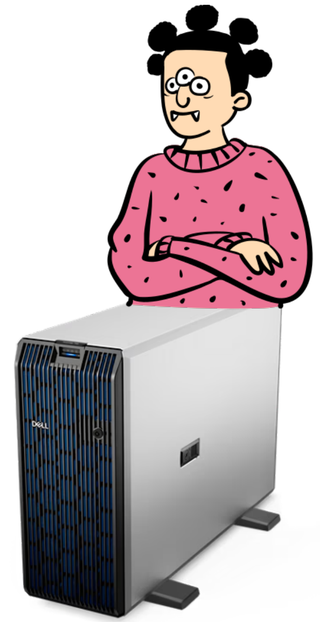


Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing

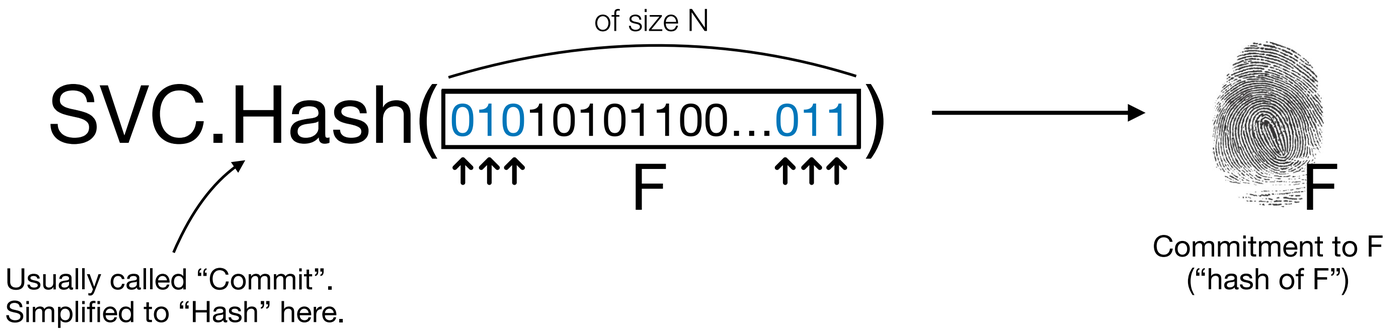


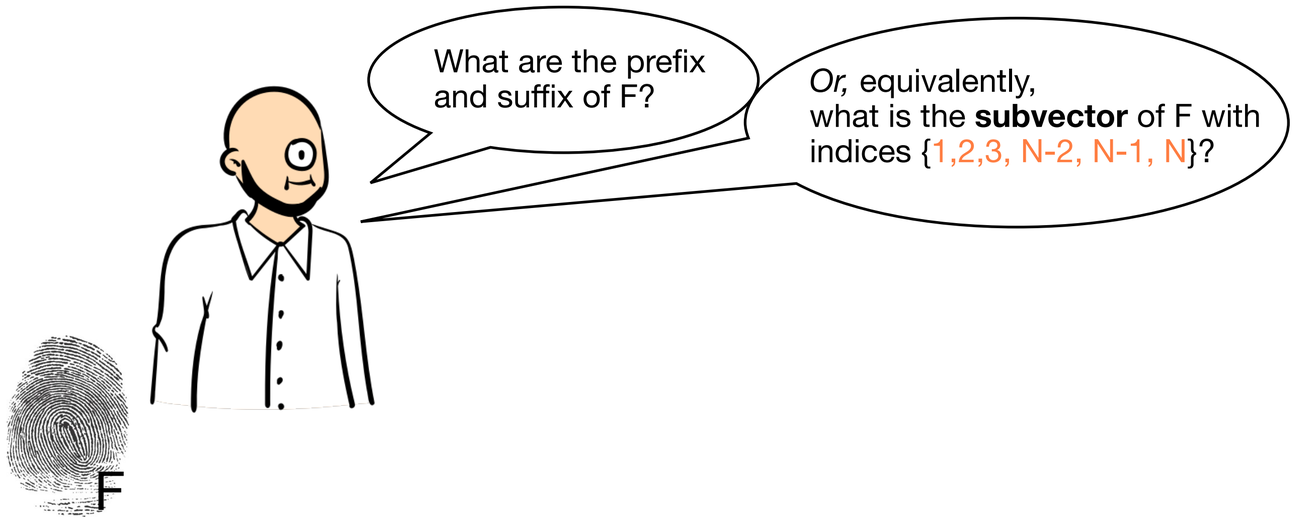


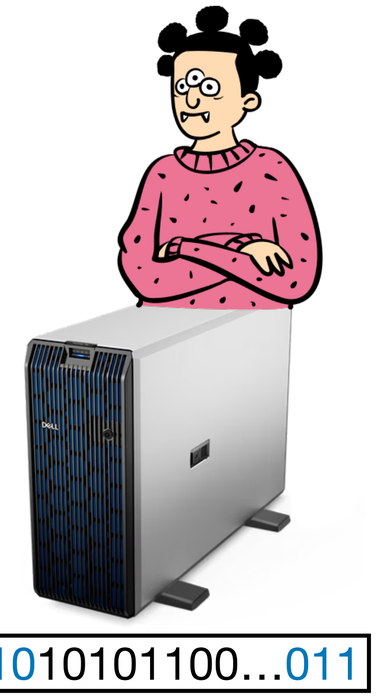


Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing







Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing

of size N

SVC.Hash(010101011100…011)

F

Usually called “Commit”.
Simplified to “Hash” here.

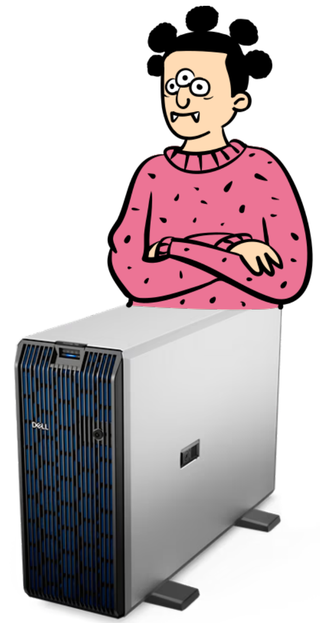
F

Commitment to F
(“hash of F”)

What are the prefix and suffix of F?

Or, equivalently, what is the **subvector** of F with indices {1,2,3, N-2, N-1, N}?

F



010101011100…011

Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing

of size N

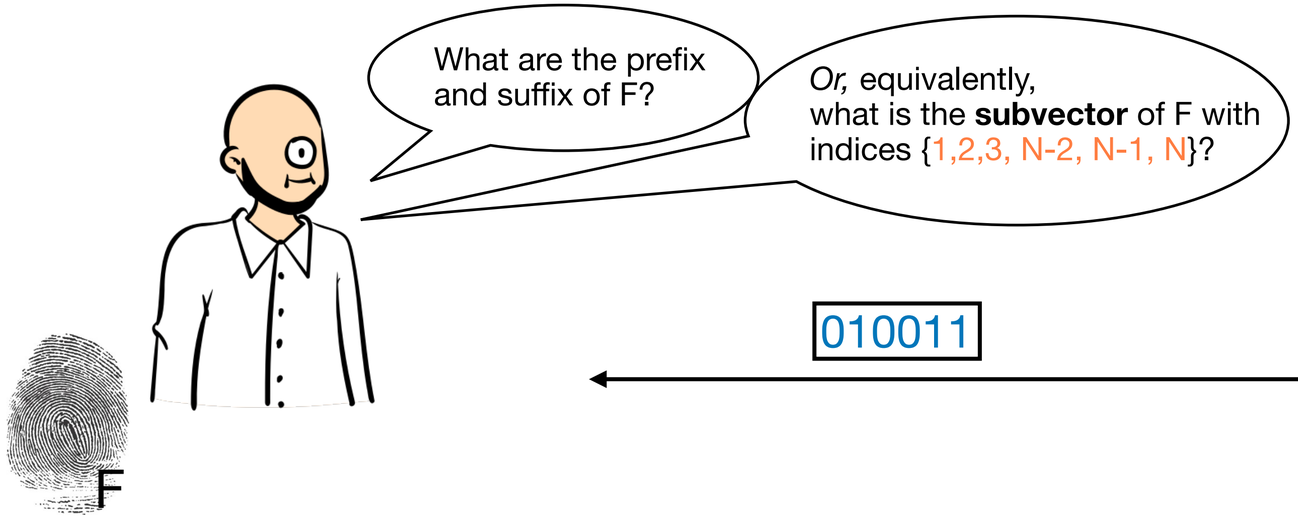
SVC.Hash(010101011100…011)

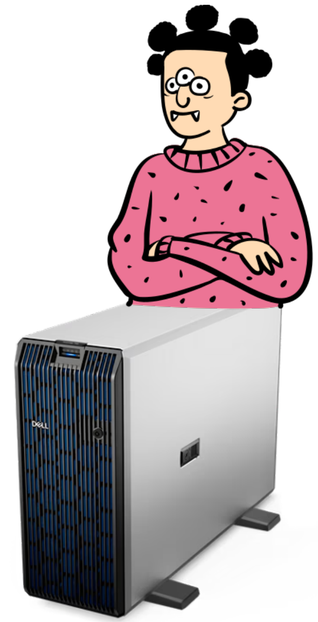
F

Usually called “Commit”.
Simplified to “Hash” here.

F

Commitment to F
(“hash of F”)

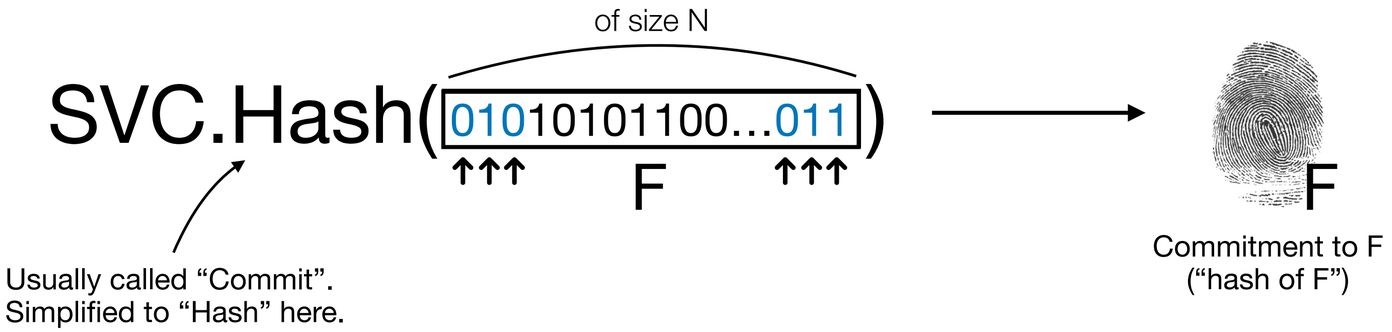


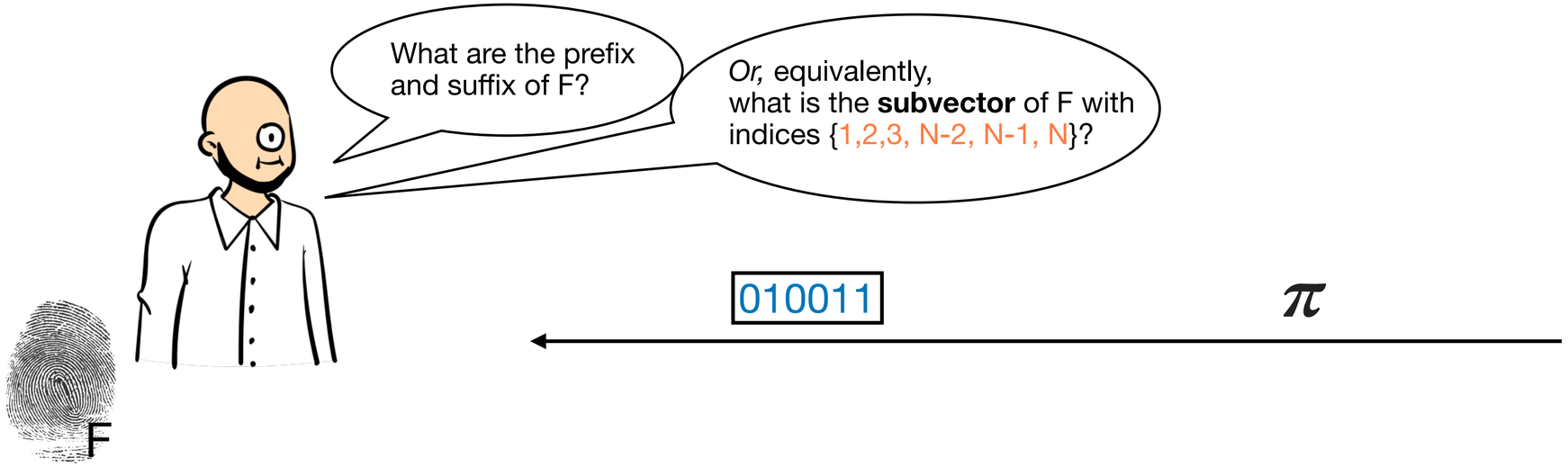


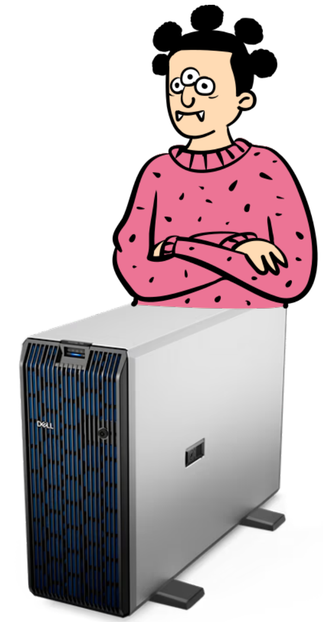
010101011100…011

Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing



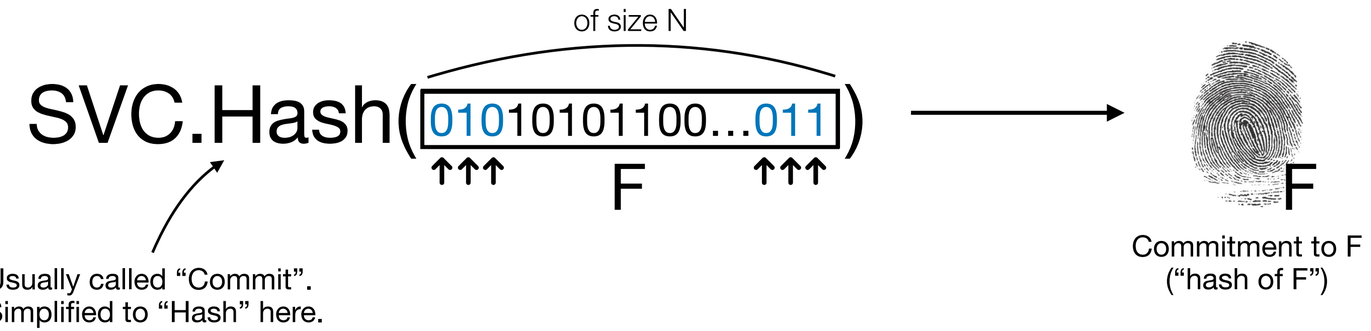


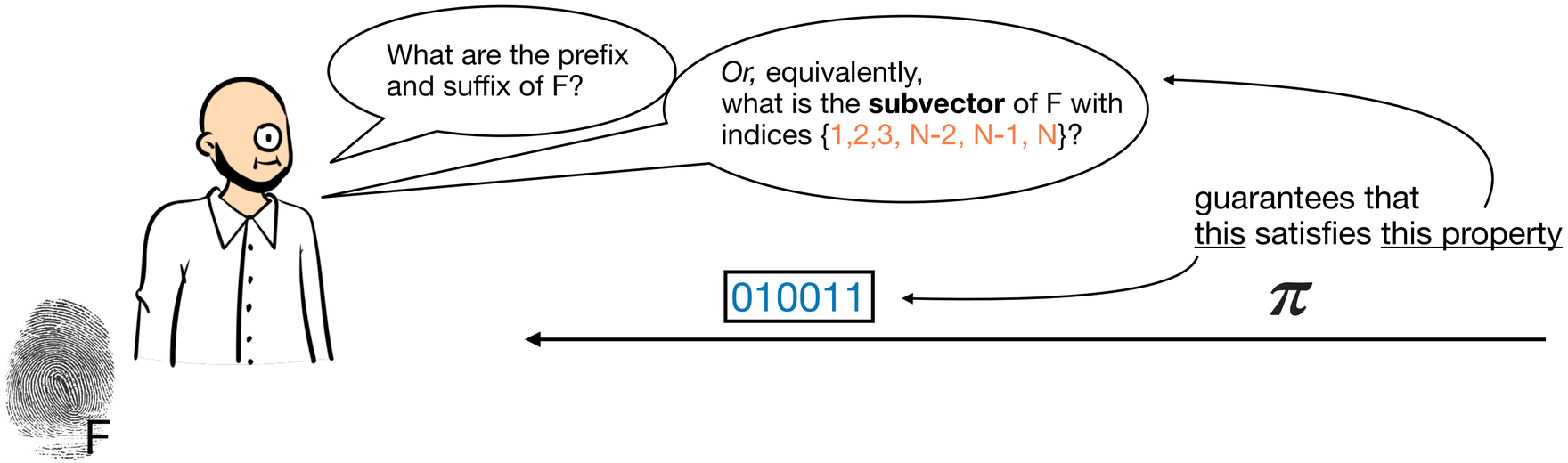


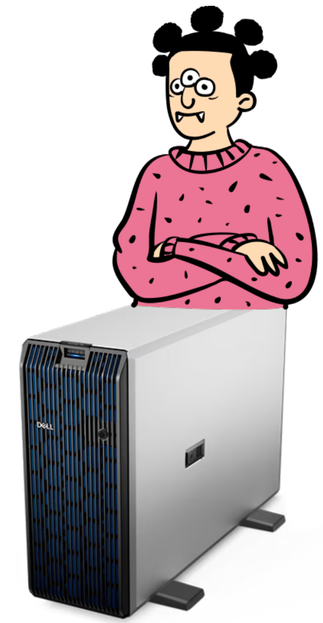
010101011100…011

Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing







010101011100...011

Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing

of size N

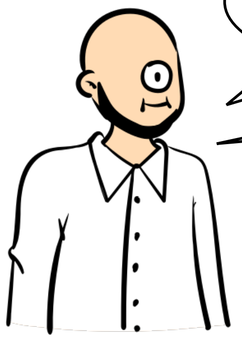
SVC.Hash(010101011100…011)

F

Usually called “Commit”.
Simplified to “Hash” here.

F

Commitment to F
(“hash of F”)



What are the prefix and suffix of F?

Or, equivalently, what is the **subvector** of F with indices {1,2,3, N-2, N-1, N}?

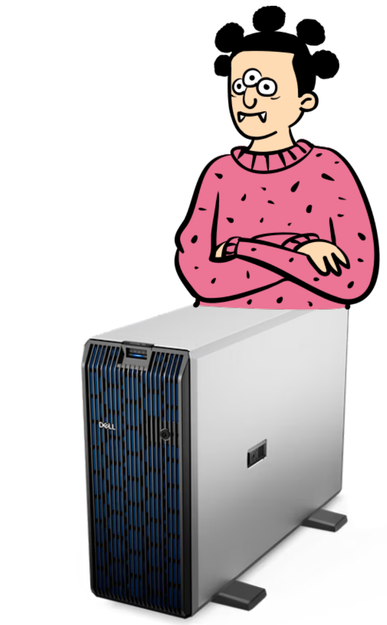
guarantees that this satisfies this property

π

010011

F

SVC.Verify(F, 010011 ,{1,2,3, N-2, N-1, N}, π)



010101011100…011

Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing

Key efficiency requirements.

of size N

SVC.Hash(010101011100…011)

F

Usually called “Commit”.
Simplified to “Hash” here.

F

Commitment to F
(“hash of F”)

What are the prefix and suffix of F?

Or, equivalently, what is the **subvector** of F with indices {1,2,3, N-2, N-1, N}?

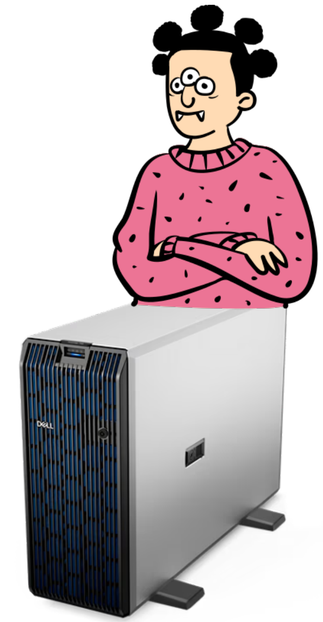
guarantees that this satisfies this property

π

010011

F

SVC.Verify(F, 010011, {1,2,3, N-2, N-1, N}, π)



010101011100…011

Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing

Key efficiency requirements.

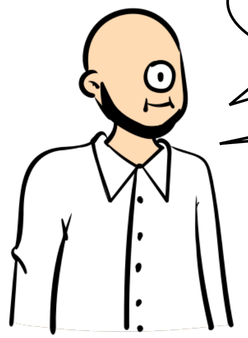
- π is short. Ideally $|\pi| = O(1)$.

of size N

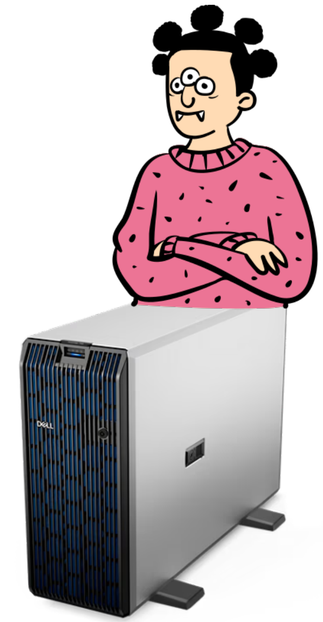
SVC.Hash(010101011100…011) → F

Usually called “Commit”.
Simplified to “Hash” here.

Commitment to F
(“hash of F”)



SVC.Verify(F, 010011, {1,2,3, N-2, N-1, N}, π)



010101011100…011

Subvector Commitments (SVC) [LY10,CF13,LM19]

A Fine-Grained Form of Hashing

Key efficiency requirements.

- π is short. Ideally $|\pi| = O(1)$.
- Time(Verify) ≈ |subvector|

of size N

SVC.Hash(010101011100…011)

F

F

Commitment to F
("hash of F")

Usually called "Commit".
Simplified to "Hash" here.

What are the prefix and suffix of F?

Or, equivalently, what is the **subvector** of F with indices {1,2,3, N-2, N-1, N}?

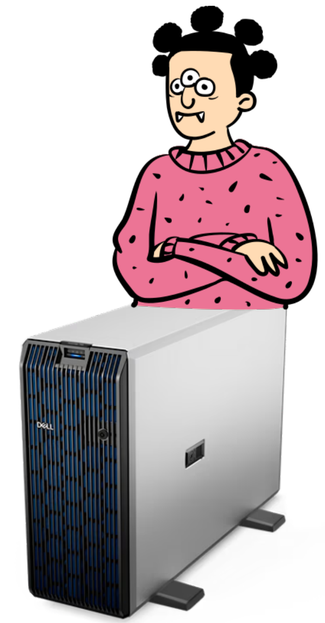
guarantees that this satisfies this property

π

010011

F

SVC.Verify( F, 010011, {1,2,3, N-2, N-1, N}, π)



010101011100…011

Applications of SVC

Applications of SVC

- Transparency Logs

Applications of SVC

- Transparency Logs
- Verifiable Databases

Applications of SVC

- Transparency Logs
- Verifiable Databases

DB

STORE

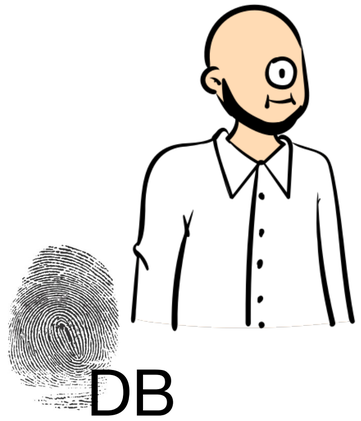
Store_key	City	Region
1	New York	**East**
2	Chicago	Central
3	Atlanta	**East**
4	Los Angeles	**West**
5	San Francisco	**West**
6	Philadelphia	**East**
.	.	.
.	.	.

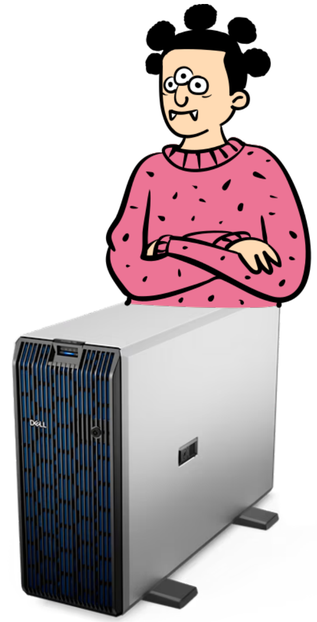
PRODUCT

Product_key	Description	Brand
1	Beautiful Girls	MKF Studios
2	Toy Story	**Wolf**
3	Sense and Sensibility	Parabuster Inc.
4	Holiday of the Year	**Wolf**
5	Pulp Fiction	MKF Studios
6	The Juror	MKF Studios
7	From Dusk Till Dawn	Parabuster Inc.
8	Hellraiser: Bloodline	Big Studios
.	.	.
.	.	.

SALES_FACT

Store_key	Product_key	Sales	**Cost**	Profit
1	6	2.39	1.15	1.24
1	2	16.7	6.91	9.79
2	7	7.16	2.75	4.40
3	2	4.77	1.84	2.93
5	3	11.93	4.59	7.34
5	1	14.31	5.51	8.80
.	.	.	.	.
.	.	.	.	.





Applications of SVC

- Transparency Logs
- Verifiable Databases

DB

STORE

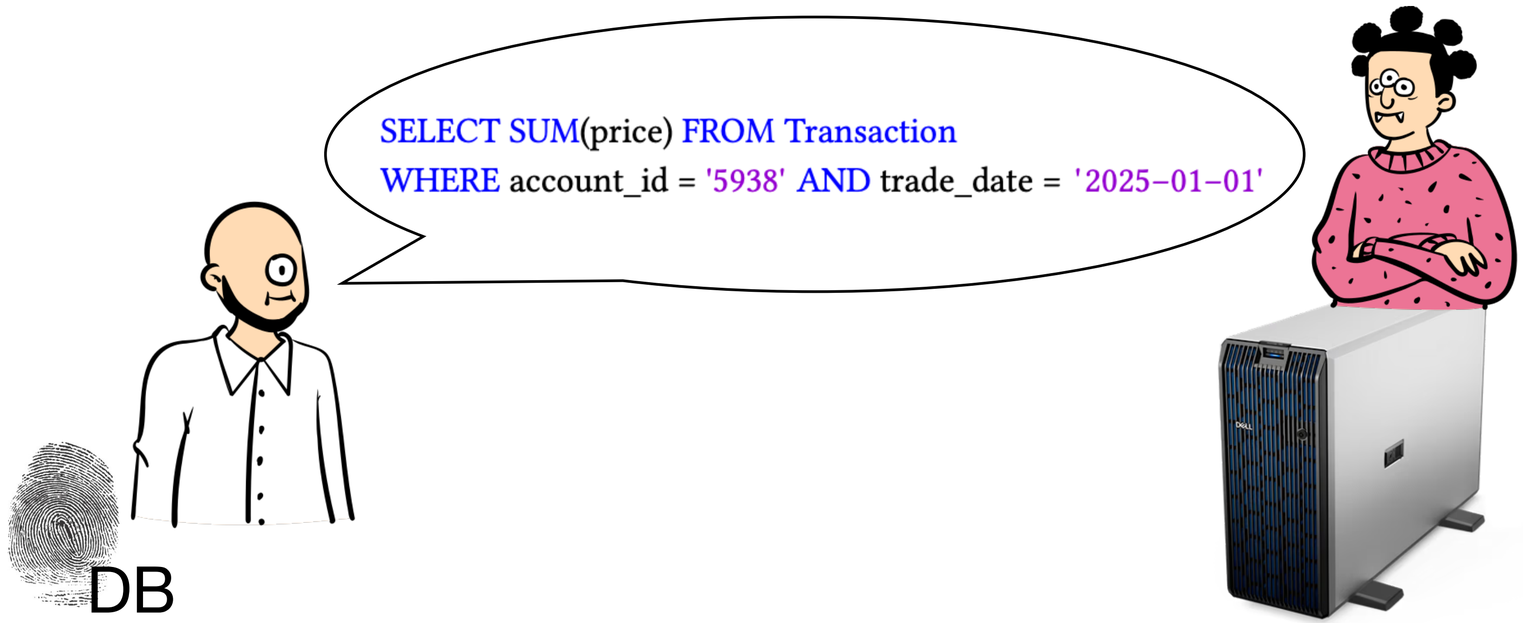
Store_key	City	Region
1	New York	**East**
2	Chicago	Central
3	Atlanta	**East**
4	Los Angeles	**West**
5	San Francisco	**West**
6	Philadelphia	**East**
.	.	.
.	.	.

PRODUCT

Product_key	Description	Brand
1	Beautiful Girls	MKF Studios
2	Toy Story	**Wolf**
3	Sense and Sensibility	Parabuster Inc.
4	Holiday of the Year	**Wolf**
5	Pulp Fiction	MKF Studios
6	The Juror	MKF Studios
7	From Dusk Till Dawn	Parabuster Inc.
8	Hellraiser: Bloodline	Big Studios
.	.	.
.	.	.

SALES_FACT

Store_key	Product_key	Sales	Cost	Profit
1	6	2.39	1.15	1.24
1	2	16.7	6.91	9.79
2	7	7.16	2.75	4.40
3	2	4.77	1.84	2.93
5	3	11.93	4.59	7.34
5	1	14.31	5.51	8.80
.	.	.	.	.
.	.	.	.	.



Applications of SVC

- Transparency Logs
- Verifiable Databases

Applications of SVC

- Transparency Logs
- Verifiable Databases
- Proofs of Space

Applications of SVC

- Transparency Logs
- Verifiable Databases
- Proofs of Space
- Compiling PCPs* into Argument Systems

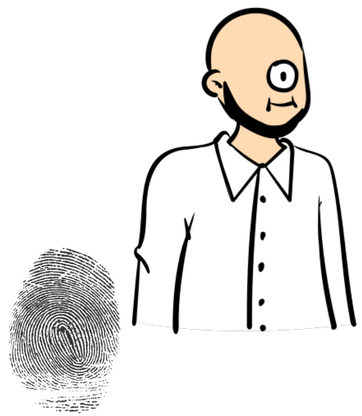
* Probabilistically Checkable Proofs

Applications of SVC

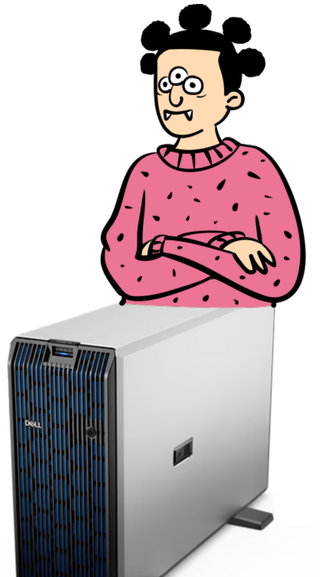
- Transparency Logs
- Verifiable Databases
- Proofs of Space
- Compiling PCPs* into Argument Systems
- “Stateless” blockchains

* Probabilistically Checkable Proofs

Focus of This Work



subvec π



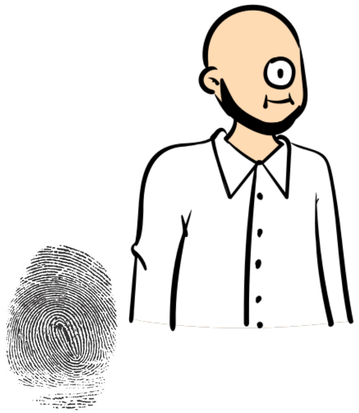
SVC.Verify($_{F}$, subvec, indices, π)

Focus of This Work

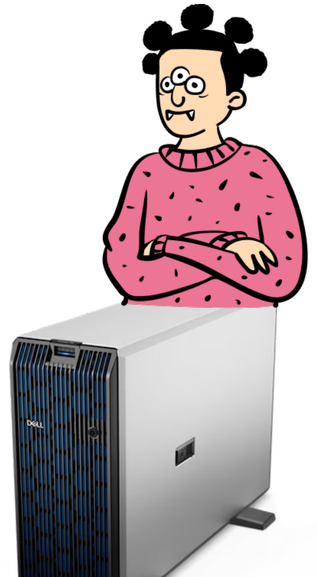
Recall:

Key efficiency requirements.

- π is short. Ideally $|\pi| = O(1)$.
- Time(Verify) ≈ |subvector|



subvec π



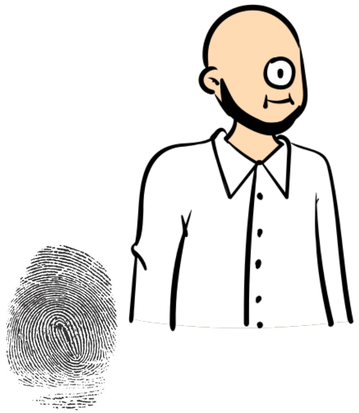
SVC.Verify(F , subvec , indices , π)

Focus of This Work

Recall:

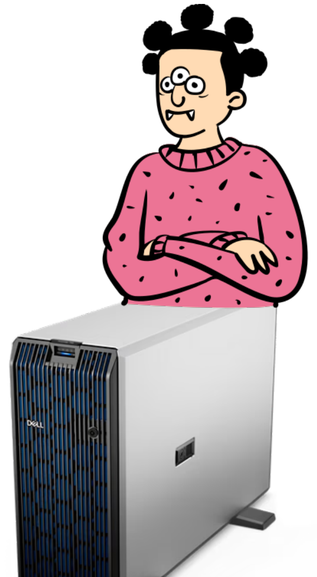
Key efficiency requirements.

- π is short. Ideally $|\pi| = O(1)$.
- Time(Verify) ≈ |subvector|



subvec π

SVC.Verify(F , subvec , indices , π)

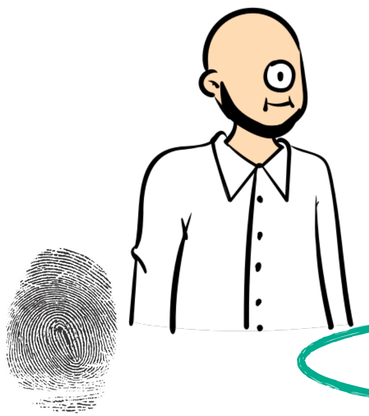


Focus of This Work

Recall:

Key efficiency requirements.

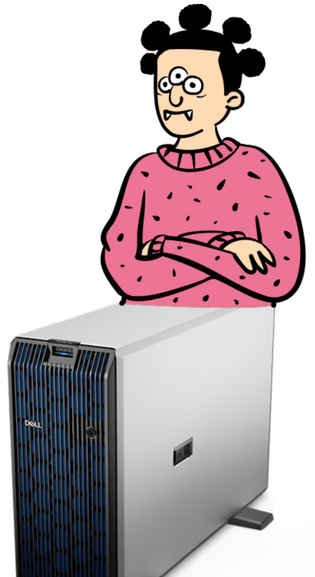
- π is short. Ideally $|\pi| = O(1)$.
- Time(Verify) ≈ |subvector|



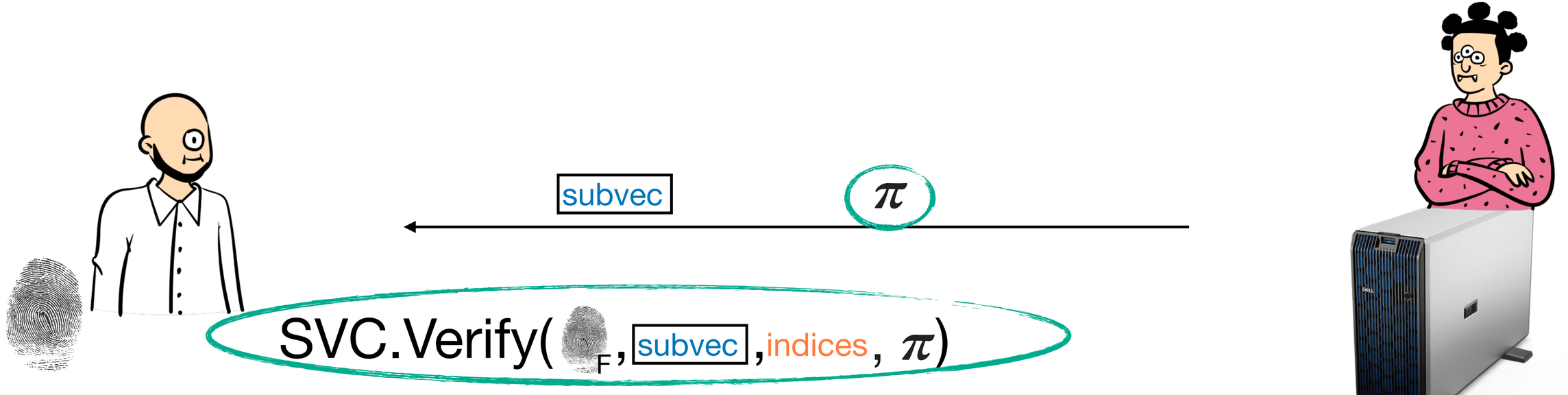
subvec

π

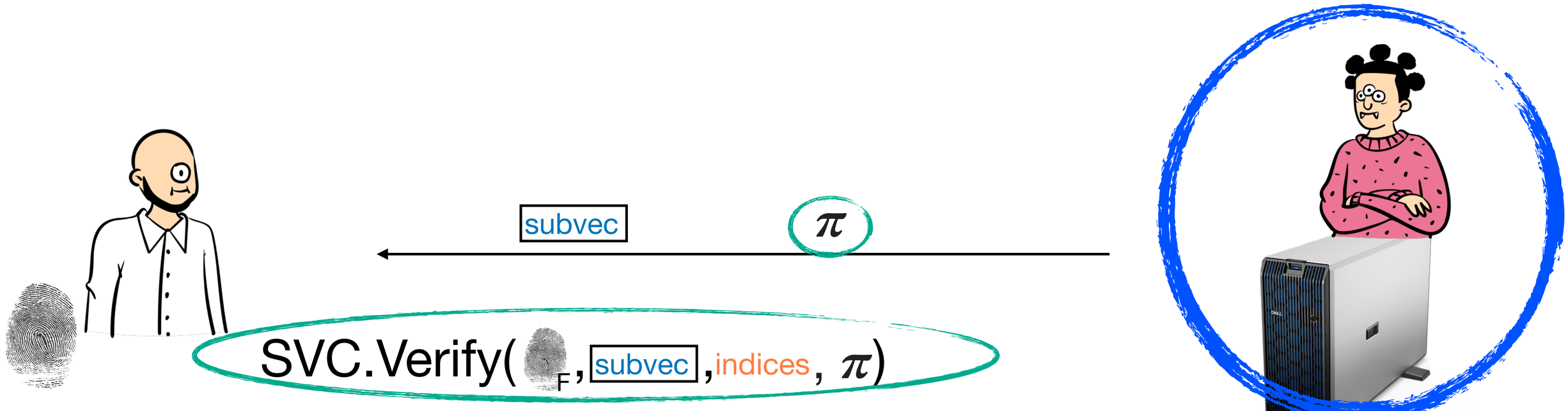
SVC.Verify(F , subvec , indices , π)



Focus of This Work

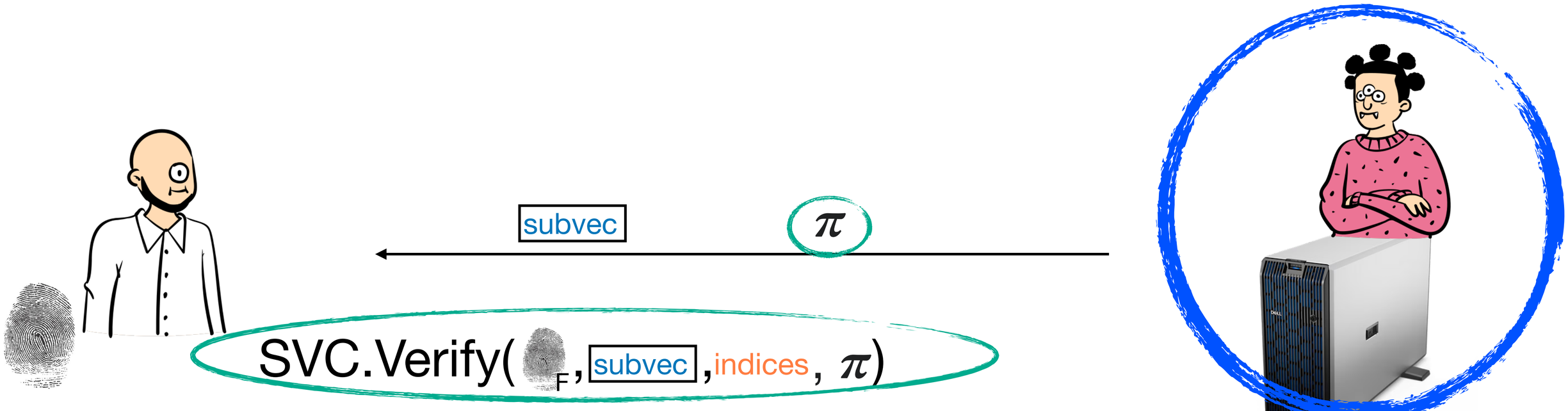


Focus of This Work

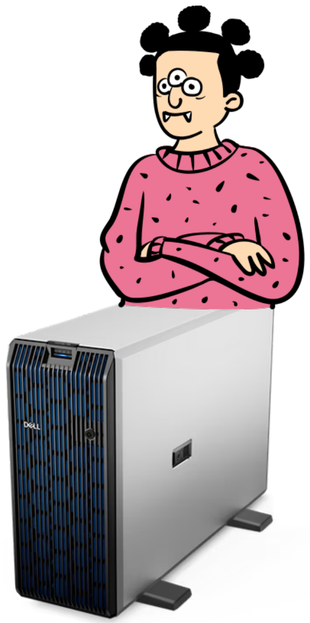


Focus of This Work

Obtaining SVCs with *optimal* proving time.



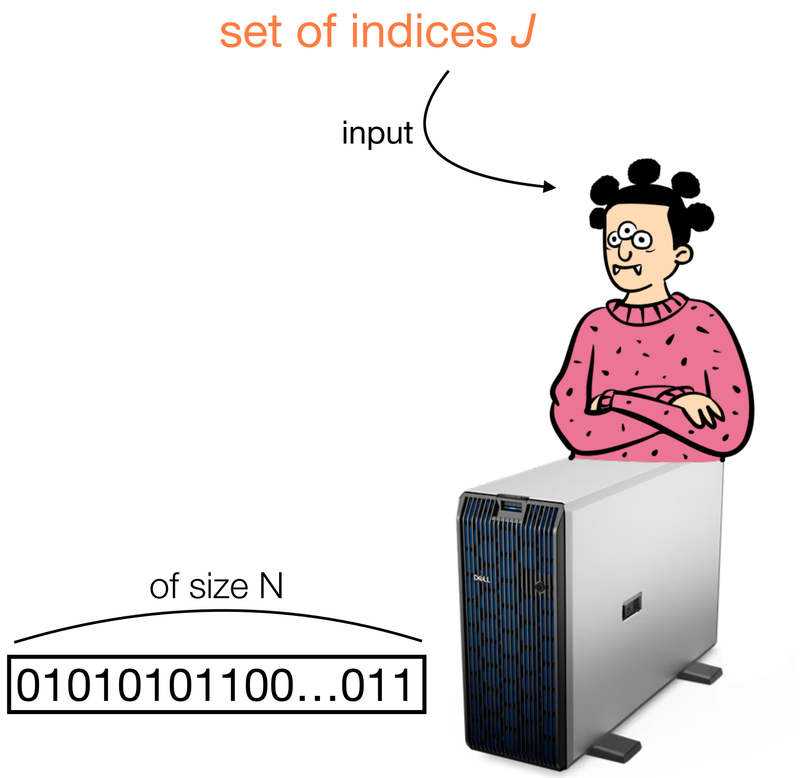
Intermezzo: What is the Optimal Proving Time in SVC?



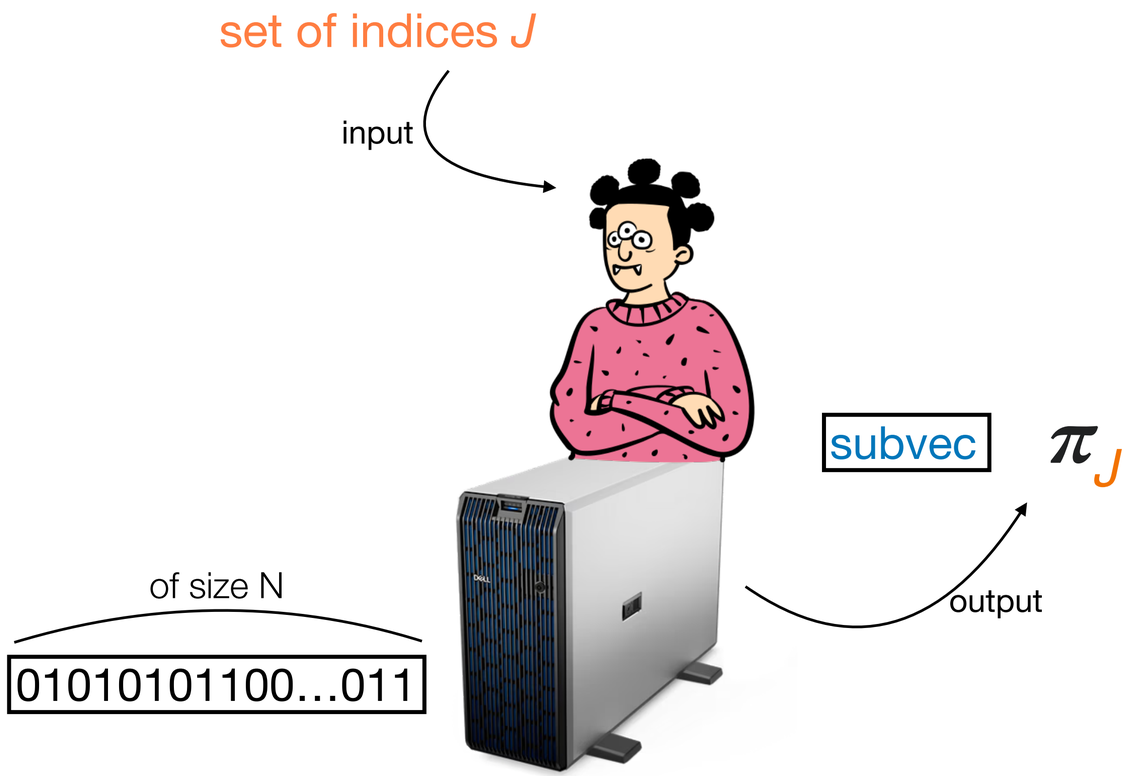
of size N

01010101100…011

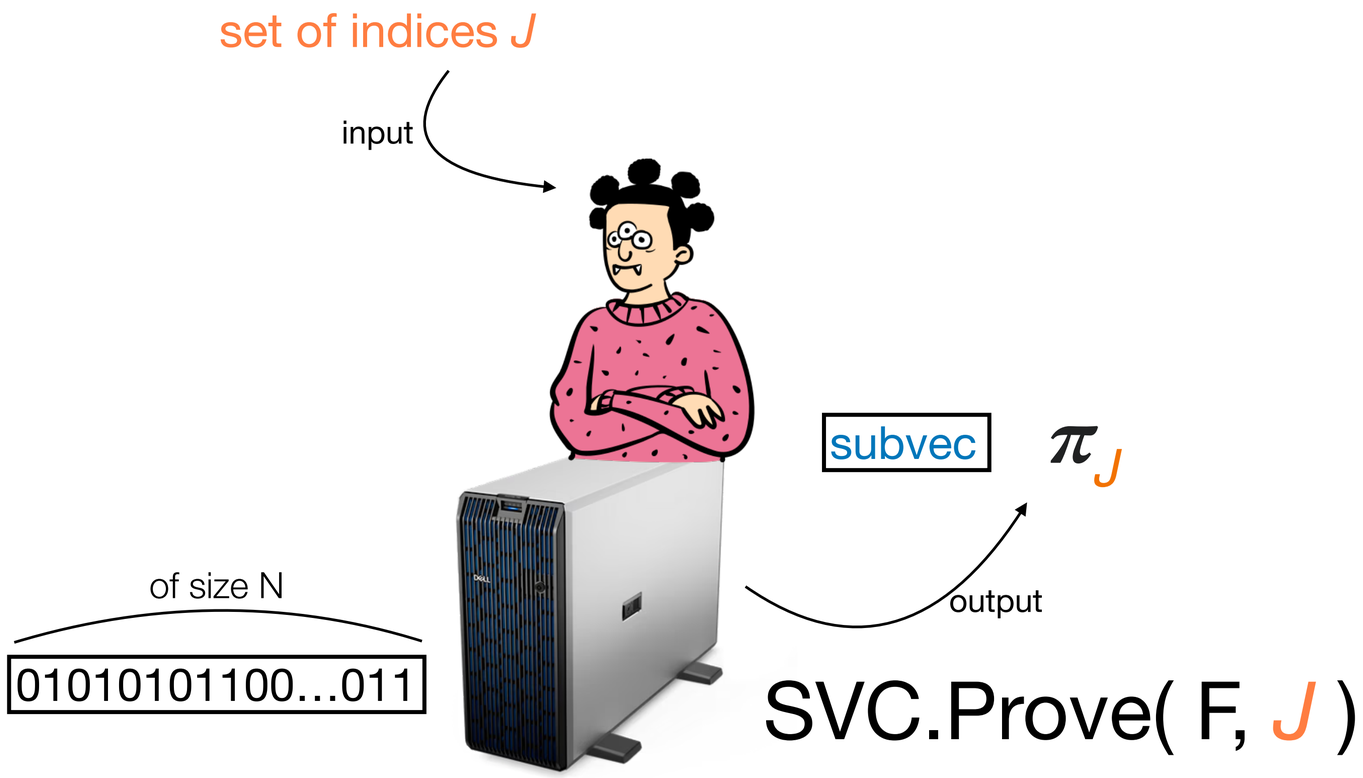
Intermezzo: What is the Optimal Proving Time in SVC?



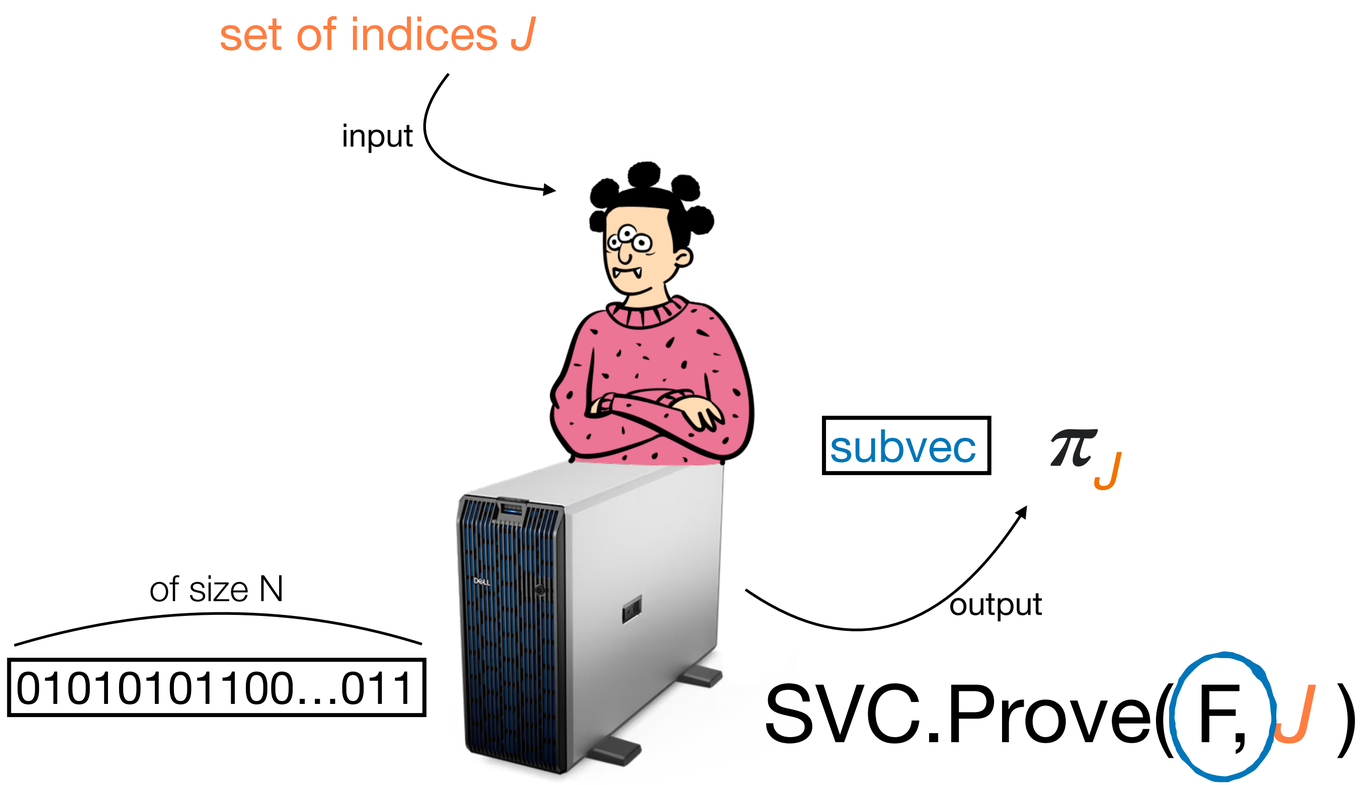
Intermezzo: What is the Optimal Proving Time in SVC?



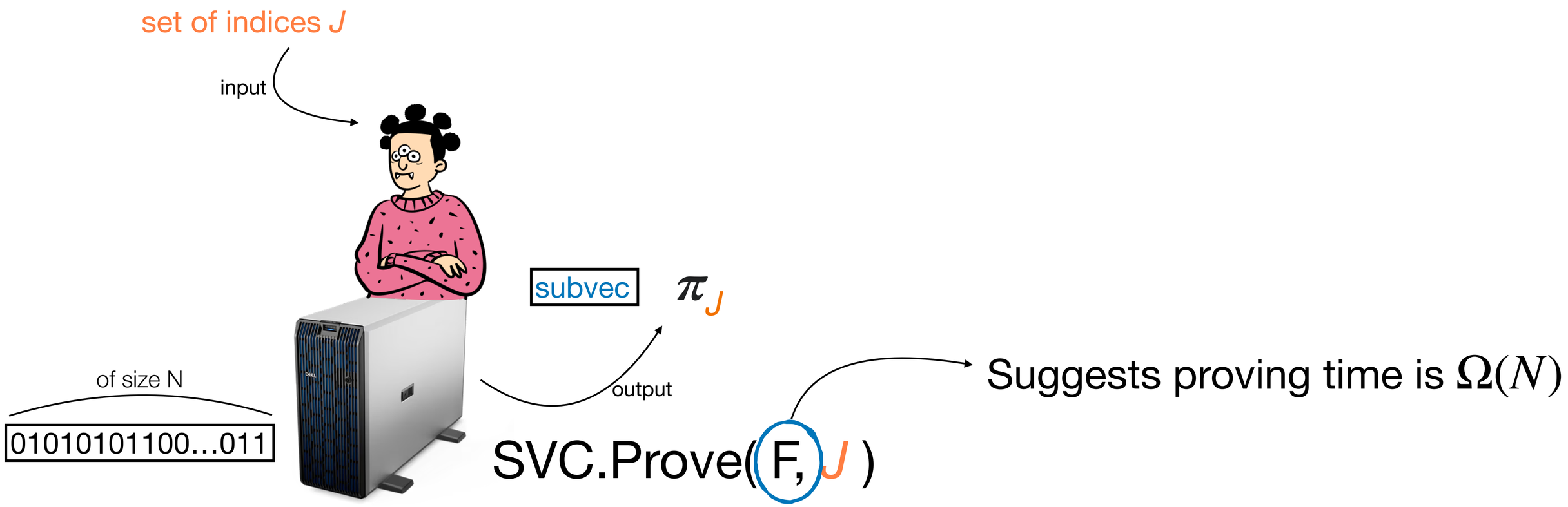
Intermezzo: What is the Optimal Proving Time in SVC?



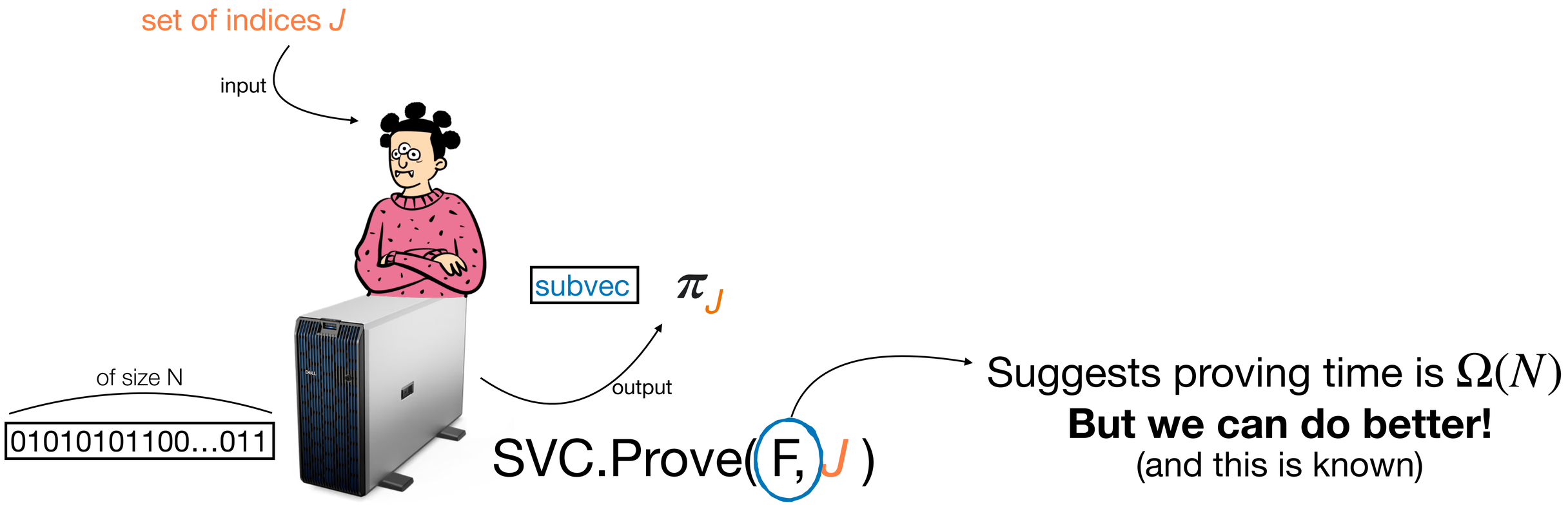
Intermezzo: What is the Optimal Proving Time in SVC?



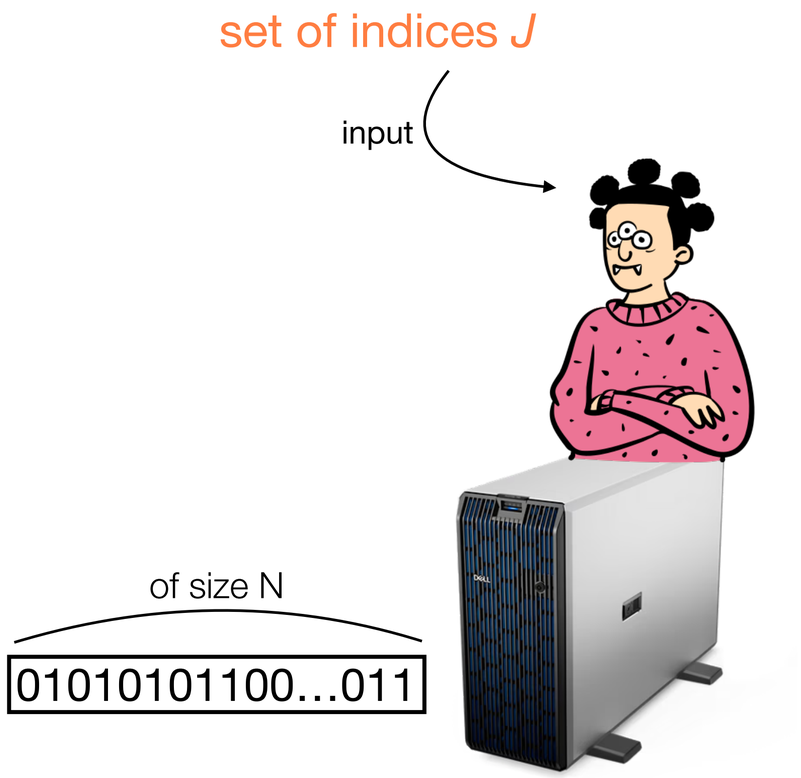
Intermezzo: What is the Optimal Proving Time in SVC?



Intermezzo: What is the Optimal Proving Time in SVC?

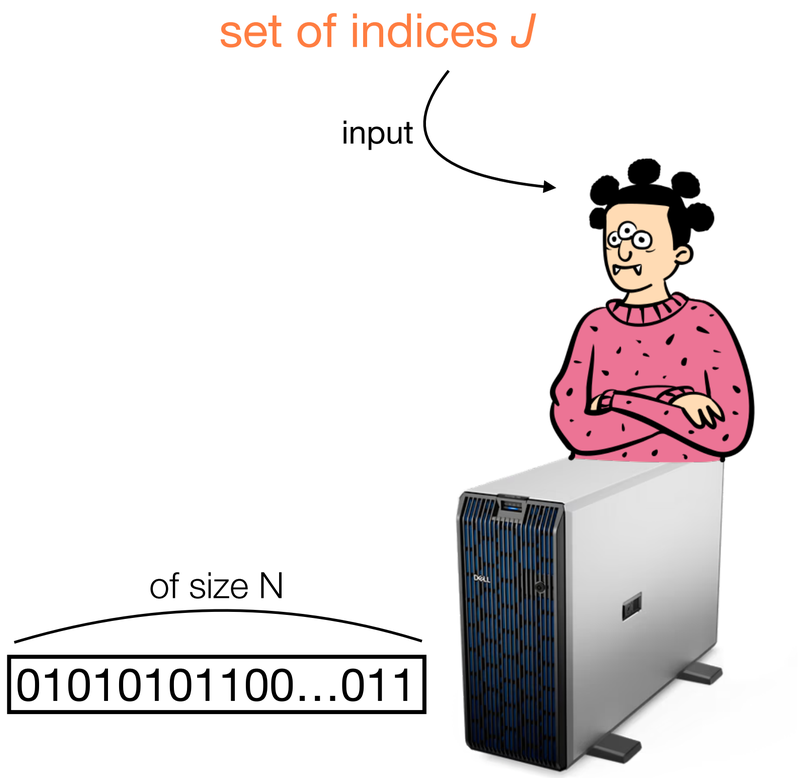


Intermezzo: What is the Optimal Proving Time in SVC?



Intermezzo: What is the Optimal Proving Time in SVC?

The preprocessing/aggregation framework:



Intermezzo: What is the Optimal Proving Time in SVC?

The preprocessing/aggregation framework:

<u>In some offline stage:</u>

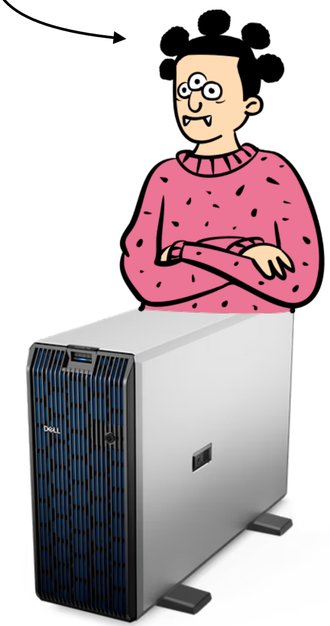
Preprocess 01010101100…011

precomputing all “singleton” proofs.

$\pi_{\{1\}}$ $\pi_{\{2\}}$ … $\pi_{\{N\}}$

set of indices *J*

input



of size N

01010101100…011

Intermezzo: What is the Optimal Proving Time in SVC?

The preprocessing/aggregation framework:

<u>In some offline stage:</u>

Preprocess 01010101100…011

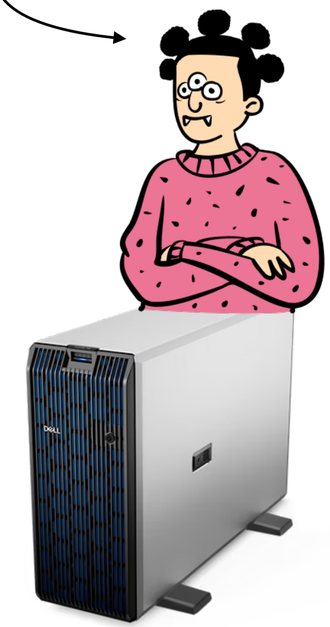
precomputing all “singleton” proofs.

$\pi_{\{1\}}$ $\pi_{\{2\}}$ … $\pi_{\{N\}}$

<u>Later:</u>

set of indices *J*

input

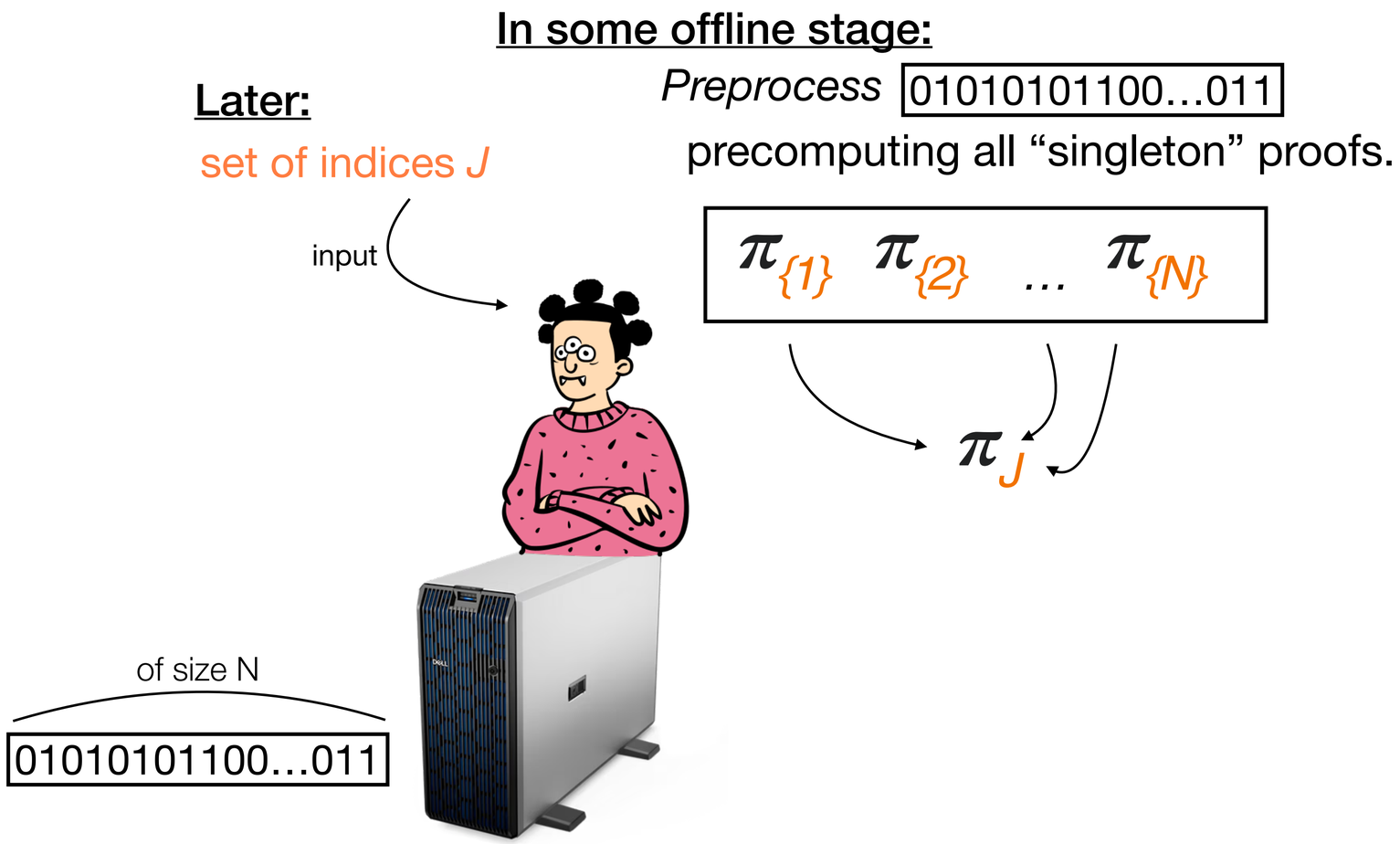


of size N

01010101100…011

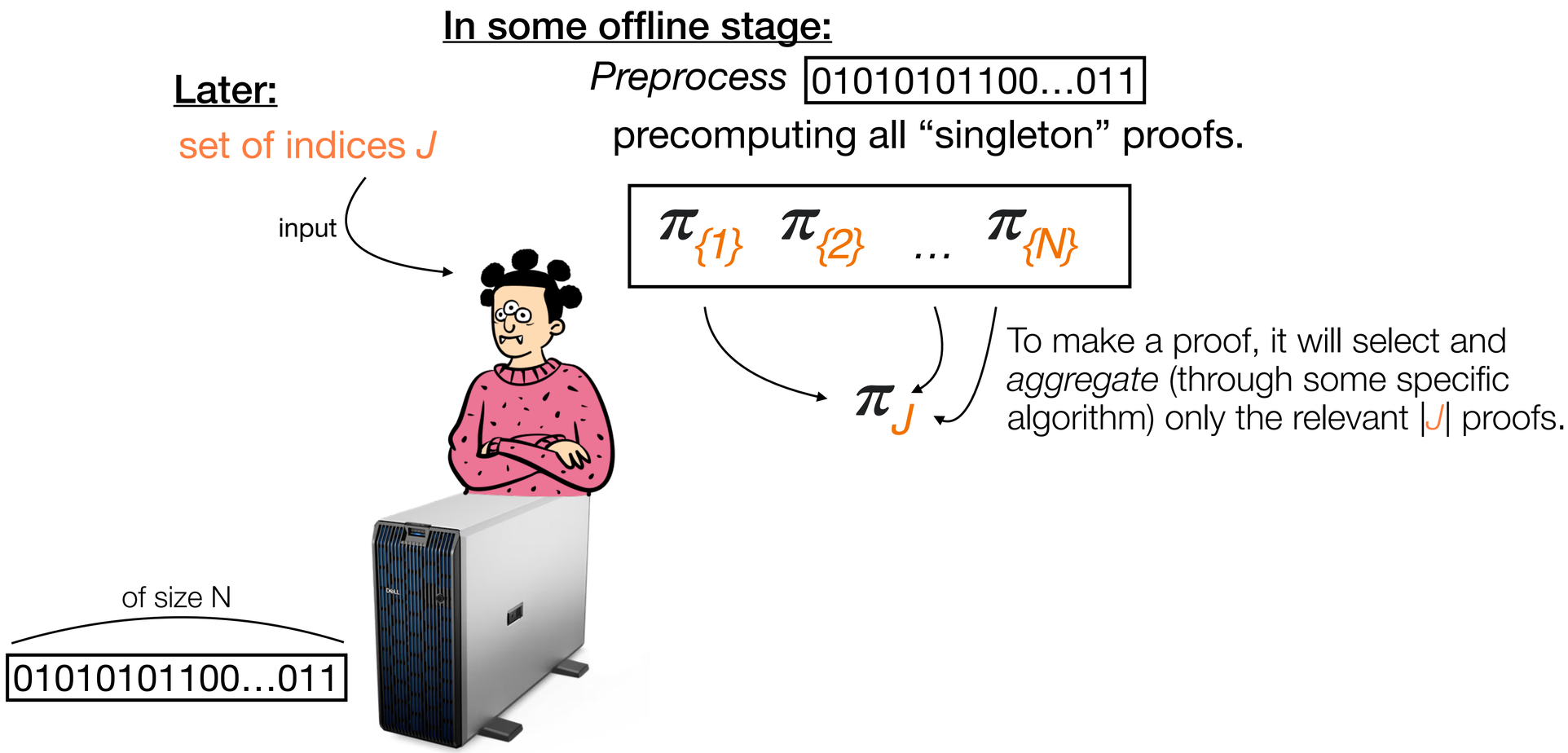
Intermezzo: What is the Optimal Proving Time in SVC?

The preprocessing/aggregation framework:



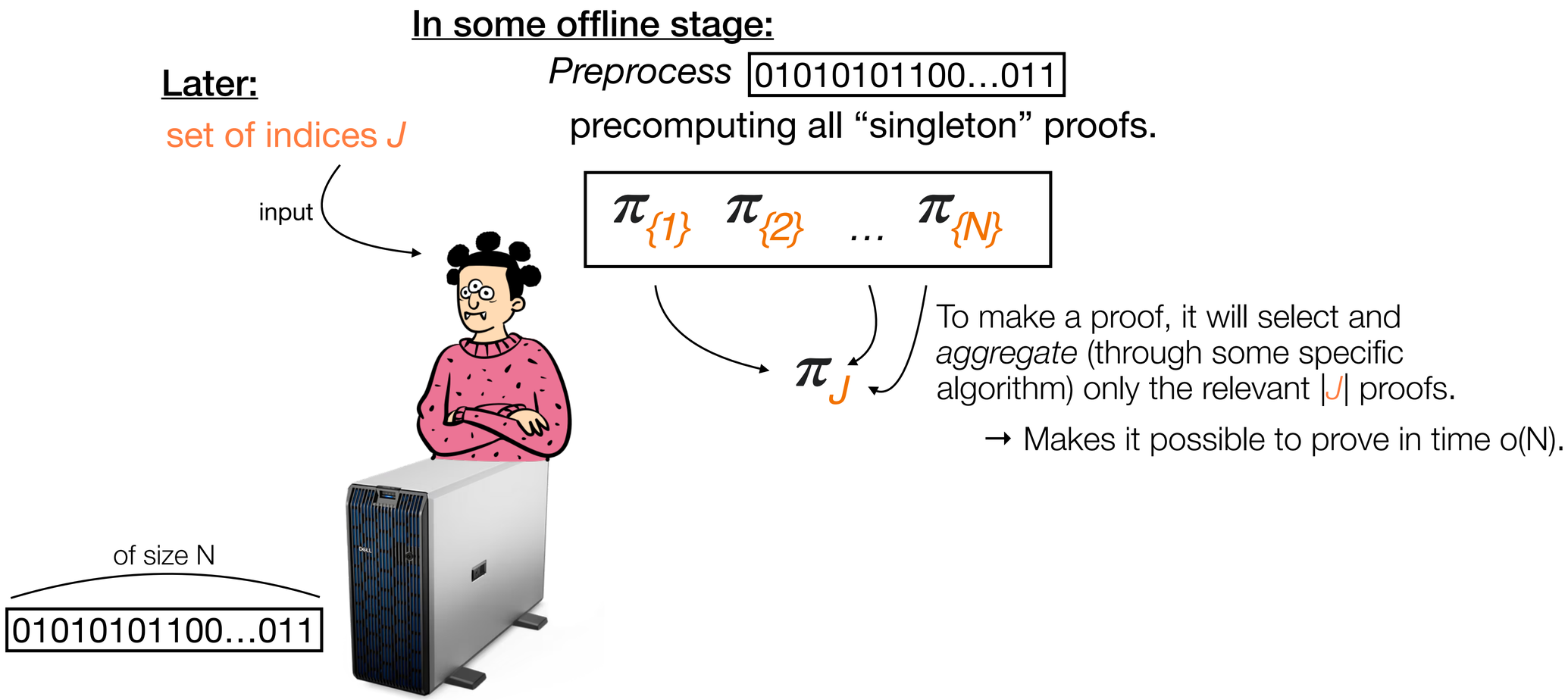
Intermezzo: What is the Optimal Proving Time in SVC?

The preprocessing/aggregation framework:



Intermezzo: What is the Optimal Proving Time in SVC?

The preprocessing/aggregation framework:



Intermezzo: What is the Optimal Proving Time in SVC?

So the optimal prover would be one strictly linear in |*J*|.

The preprocessing/aggregation framework:

<u>In some offline stage:</u>

Preprocess 01010101100…011

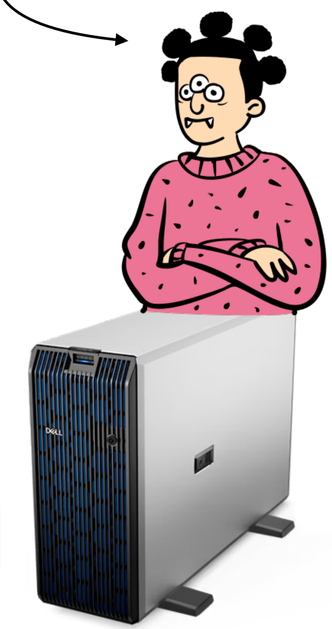
precomputing all “singleton” proofs.

$\pi_{\{1\}}$ $\pi_{\{2\}}$ … $\pi_{\{N\}}$

<u>Later:</u>

set of indices *J*

input



of size N

01010101100…011

π_J

To make a proof, it will select and *aggregate* (through some specific algorithm) only the relevant |*J*| proofs.

→ Makes it possible to prove in time o(N).

Intermezzo: What is the Optimal Proving Time in SVC?

So the optimal prover would be one strictly linear in |J|.

Yet, no prior construction has this efficiency profile.

The preprocessing/aggregation framework:

<u>In some offline stage:</u>

Preprocess 01010101100…011

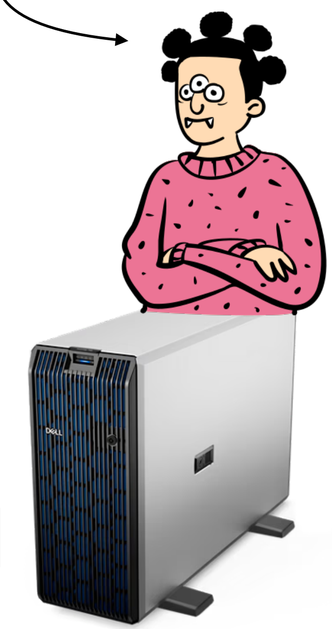
precomputing all “singleton” proofs.

$\pi_{\{1\}}$ $\pi_{\{2\}}$ … $\pi_{\{N\}}$

<u>Later:</u>

set of indices J

input



of size N

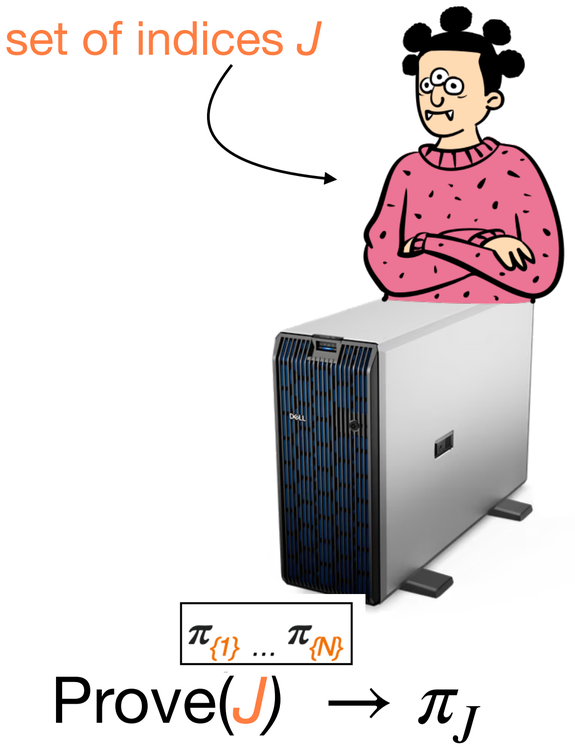
01010101100…011

π_J

To make a proof, it will select and *aggregate* (through some specific algorithm) only the relevant |J| proofs.

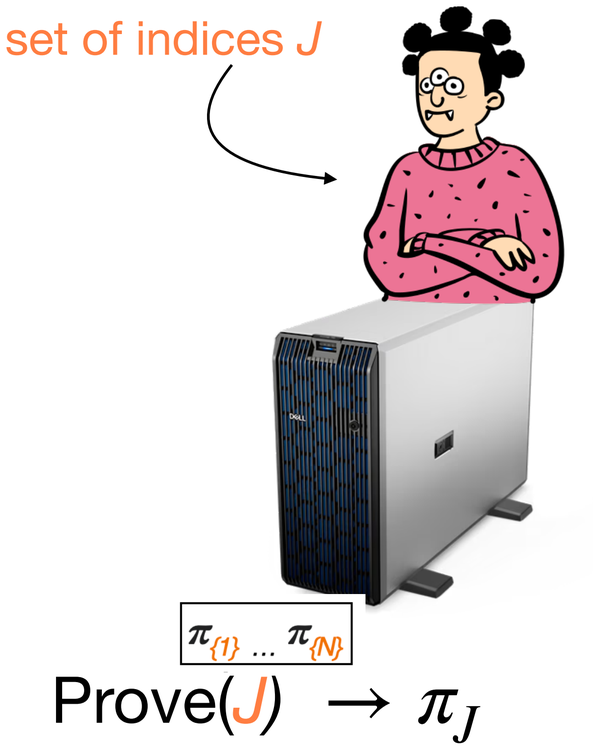
→ Makes it possible to prove in time o(N).

Main Result of This Work



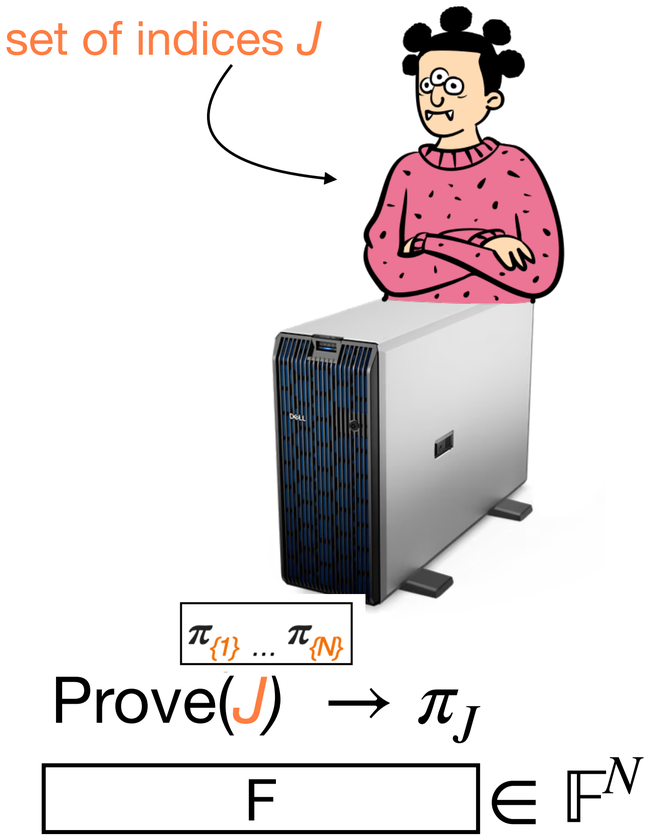
Main Result of This Work

First SVC with $\Theta(|J|)$ proving time.



Main Result of This Work

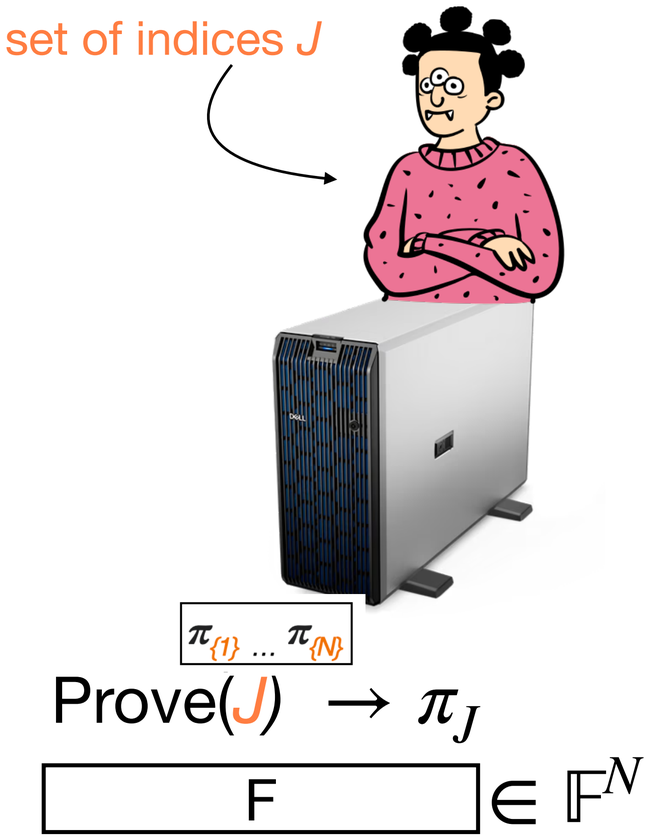
First SVC with $\Theta(|J|)$ proving time.



Main Result of This Work

scalar ops in $\mathbb{F}$

First SVC with $\Theta(|J|)$ proving time.

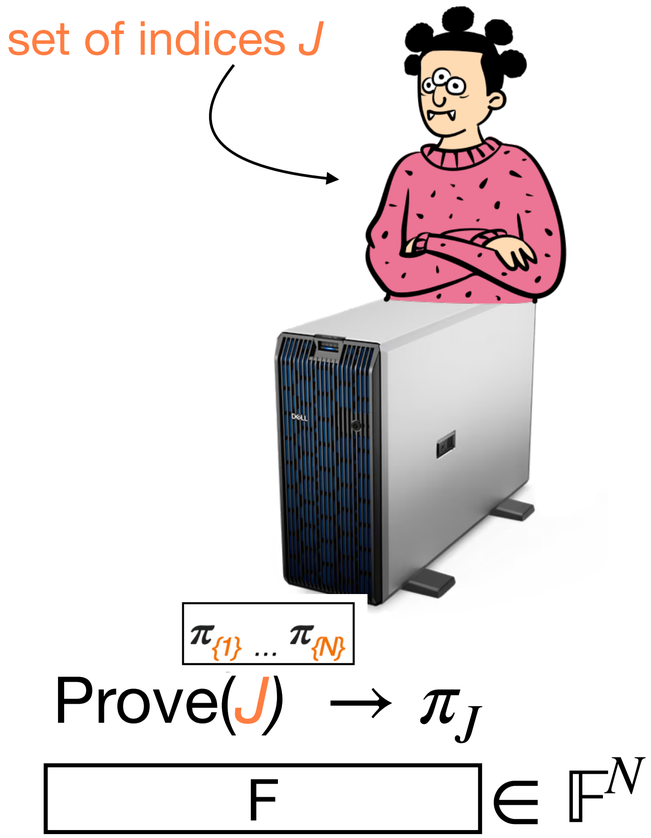


Main Result of This Work

scalar ops in $\mathbb{F}$

First SVC with $\Theta(|J|)$ proving time.

(this is without sacrificing bandwidth*: it has O(1) commitment and proof size)



*nor verification time—more in a second.

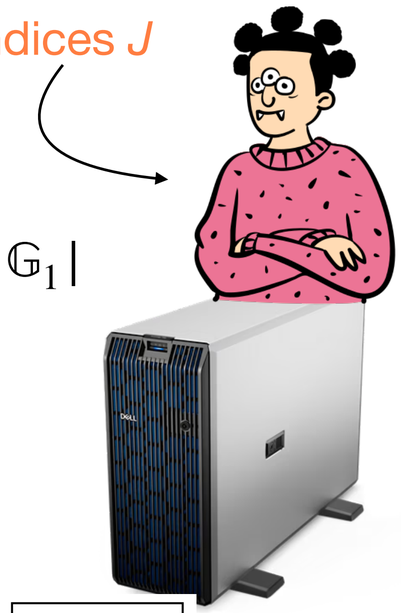
Main Result of This Work

scalar ops in $\mathbb{F}$

First SVC with $\Theta(|J|)$ proving time.

(this is without sacrificing bandwidth*: it has O(1) commitment and proof size)

$|\mathsf{cm}| = |\pi| = 1\,|\mathbb{G}_1|$

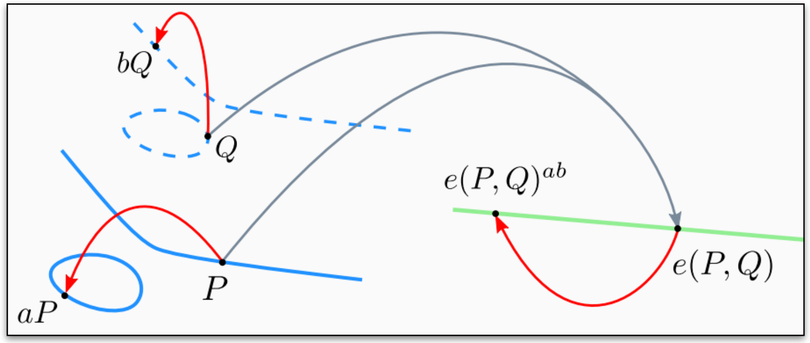


$\mathsf{Prove}(J) \rightarrow \pi_J$

F $\in \mathbb{F}^N$

*nor verification time—more in a second.

Main Result of This Work



Our setting is elliptic curves with bilinear pairings. [pic credit: Diego Aranha]

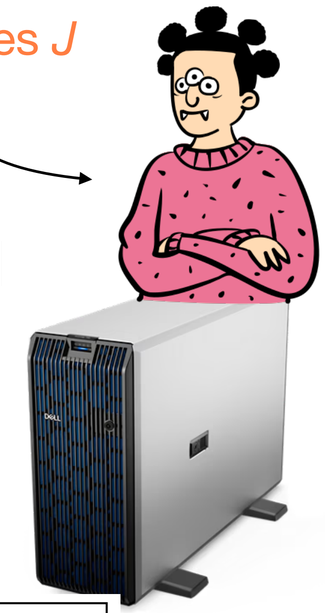
scalar ops in $\mathbb{F}$

First SVC with $\Theta(|J|)$ proving time.

(this is without sacrificing bandwidth*: it has O(1) commitment and proof size)

$|\mathsf{cm}| = |\pi| = 1\,|\mathbb{G}_1|$

set of indices J



$\pi_{\{1\}} \ldots \pi_{\{N\}}$

$\mathsf{Prove}(J) \rightarrow \pi_J$

F $\in \mathbb{F}^N$

*nor verification time—more in a second.

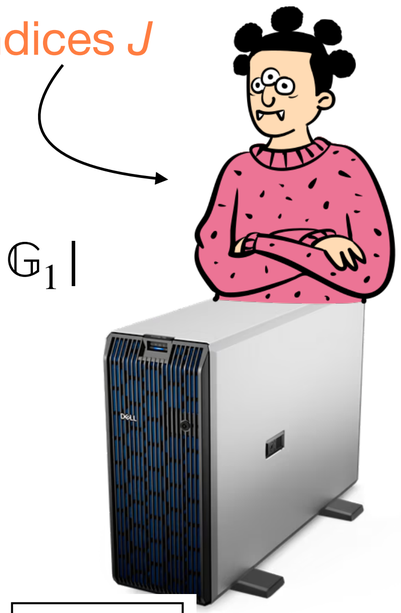
Main Result of This Work

scalar ops in $\mathbb{F}$

First SVC with $\Theta(|J|)$ proving time.

(this is without sacrificing bandwidth*: it has O(1) commitment and proof size)

$|\mathsf{cm}| = |\pi| = 1\,|\mathbb{G}_1|$



$\mathsf{Prove}(J) \rightarrow \pi_J$

F $\in \mathbb{F}^N$

*nor verification time—more in a second.

Main Result of This Work

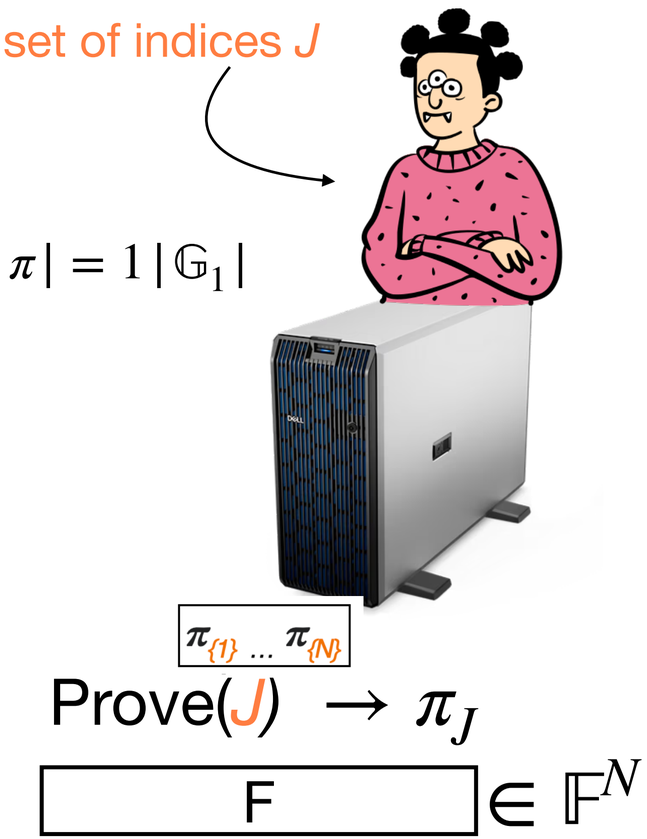
scalar ops in $\mathbb{F}$

First SVC with $\Theta(|J|)$ proving time.

(this is without sacrificing bandwidth*: it has O(1) commitment and proof size)

$|\mathsf{cm}| = |\pi| = 1\,|\mathbb{G}_1|$

"Cute asymptotics you got there—but why would I care?"



*nor verification time—more in a second.

Main Result of This Work

scalar ops in $\mathbb{F}$

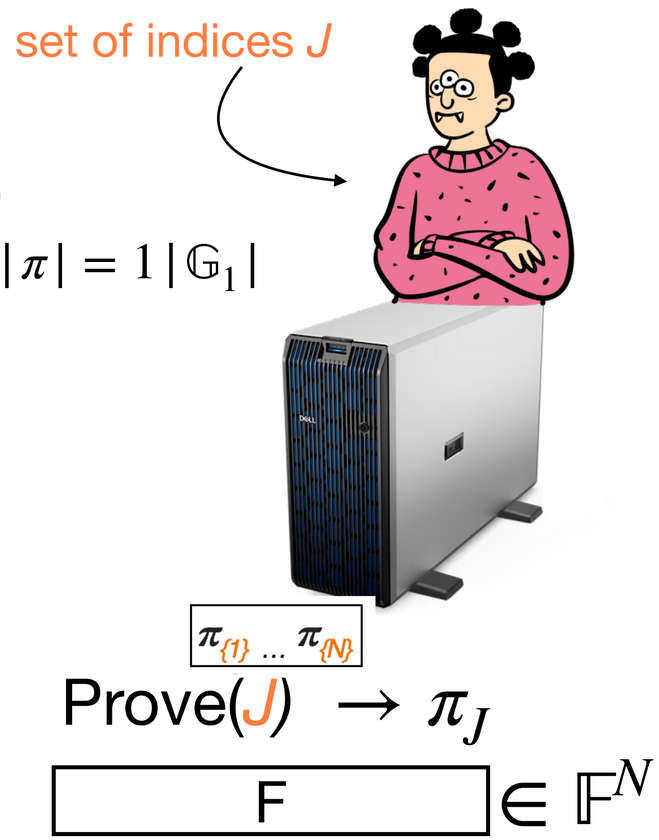
First SVC with $\Theta(|J|)$ proving time.

(this is without sacrificing bandwidth*: it has O(1) commitment and proof size)

$|\mathsf{cm}| = |\pi| = 1\,|\mathbb{G}_1|$

"Cute asymptotics you got there—but why would I care?"

- **Concretely faster**



*nor verification time—more in a second.

Main Result of This Work

scalar ops in $\mathbb{F}$

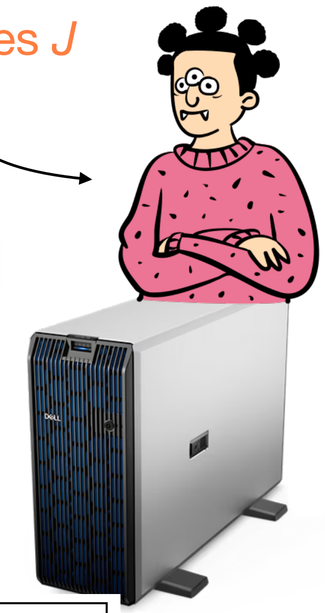
First SVC with $\Theta(|J|)$ proving time.

(this is without sacrificing bandwidth*: it has O(1) commitment and proof size)

$|\mathsf{cm}| = |\pi| = 1\,|\mathbb{G}_1|$

"Cute asymptotics you got there—but why would I care?"

- **Concretely faster**
  - 2—3 *orders-of-magnitude* speedups for Prove (similar effects for Verify)



*nor verification time—more in a second.

Main Result of This Work

scalar ops in $\mathbb{F}$

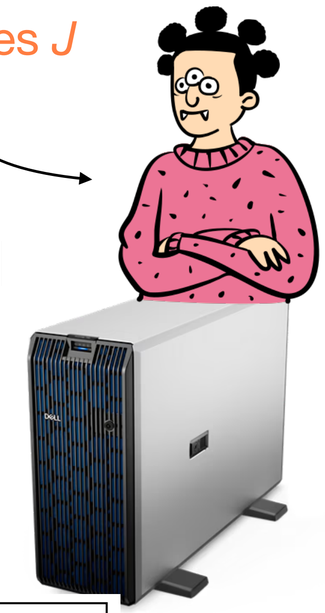
First SVC with $\Theta(|J|)$ proving time.

(this is without sacrificing bandwidth*: it has O(1) commitment and proof size)

$|\mathsf{cm}| = |\pi| = 1\,|\mathbb{G}_1|$

"Cute asymptotics you got there—but why would I care?"

- **Concretely faster**
  - 2—3 *orders-of-magnitude* speedups for Prove (similar effects for Verify)
- **No random oracle for proving**



*nor verification time—more in a second.

Main Result of This Work

scalar ops in $\mathbb{F}$

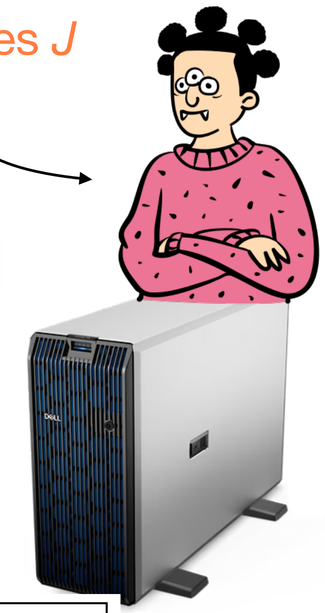
First SVC with $\Theta(|J|)$ proving time.

(this is without sacrificing bandwidth*: it has O(1) commitment and proof size)

$|\mathsf{cm}| = |\pi| = 1\,|\mathbb{G}_1|$

"Cute asymptotics you got there—but why would I care?"

- **Concretely faster**
  - 2—3 *orders-of-magnitude* speedups for Prove (similar effects for Verify)
- **No random oracle for proving**
  - ⇒ more efficient composition with other cryptographic proofs



*nor verification time—more in a second.

Main Result of This Work

scalar ops in $\mathbb{F}$

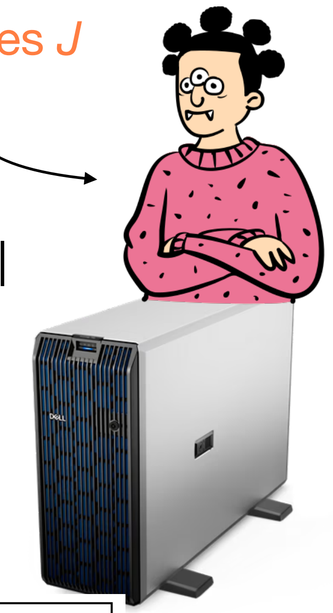
First SVC with $\Theta(|J|)$ proving time.

(this is without sacrificing bandwidth*: it has O(1) commitment and proof size)

$|\mathsf{cm}| = |\pi| = 1\,|\mathbb{G}_1|$

"Cute asymptotics you got there—but why would I care?"

- **Concretely faster**
  - 2—3 *orders-of-magnitude* speedups for Prove (similar effects for Verify)
- **No random oracle for proving**
  - ⇒ more efficient composition with other cryptographic proofs
- **Fully associative proofs**



*nor verification time—more in a second.

Main Result of This Work

scalar ops in $\mathbb{F}$

First SVC with $\Theta(|J|)$ proving time.

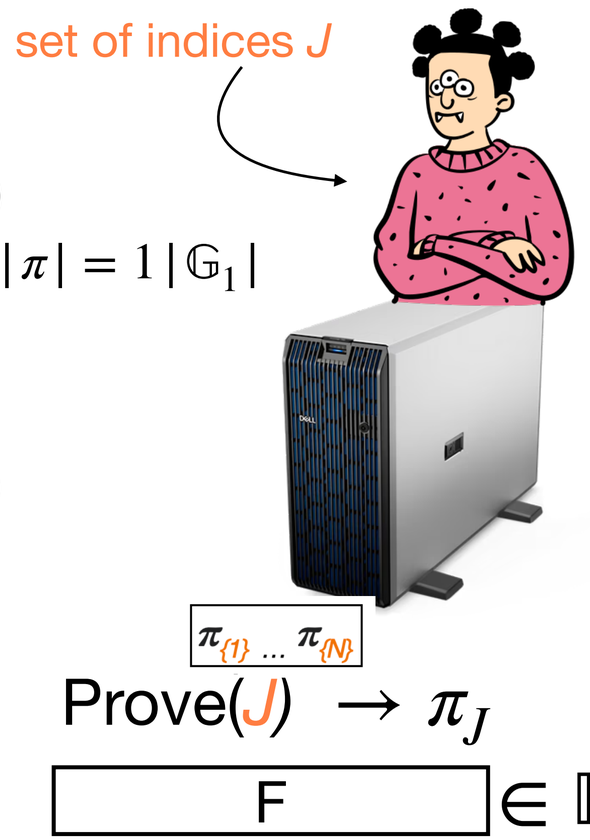
(this is without sacrificing bandwidth*: it has O(1) commitment and proof size)

$|\mathsf{cm}| = |\pi| = 1\,|\mathbb{G}_1|$

"Cute asymptotics you got there—but why would I care?"

- **Concretely faster**
  - 2—3 *orders-of-magnitude* speedups for Prove (similar effects for Verify)
- **No random oracle for proving**
  - ⇒ more efficient composition with other cryptographic proofs
- **Fully associative proofs**

$$\pi_{I \cup J} = \pi_I + \pi_J$$



*nor verification time—more in a second.

Main Result of This Work

scalar ops in $\mathbb{F}$

First SVC with $\Theta(|J|)$ proving time.

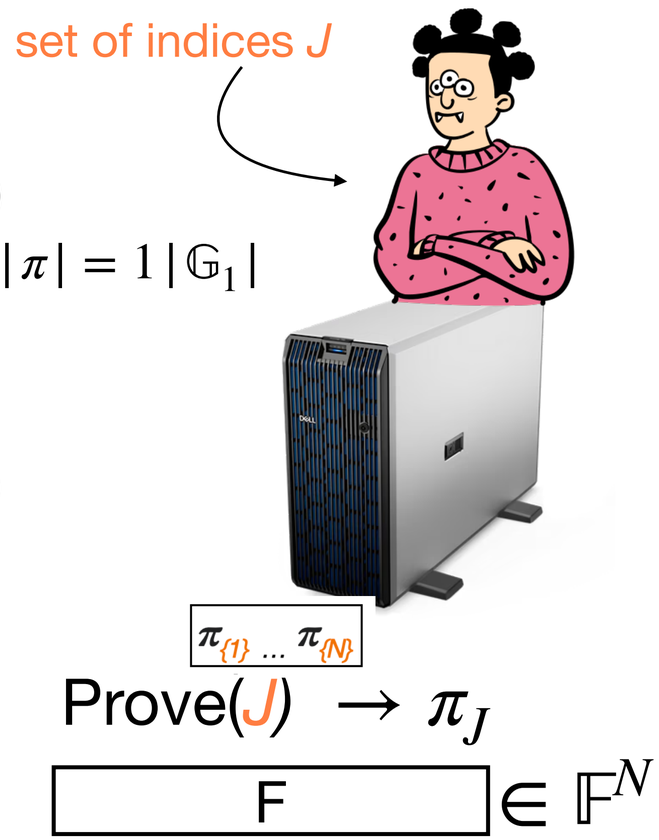
(this is without sacrificing bandwidth*: it has O(1) commitment and proof size)

$|\mathsf{cm}| = |\pi| = 1\,|\mathbb{G}_1|$

"Cute asymptotics you got there—but why would I care?"

- **Concretely faster**
  - 2—3 *orders-of-magnitude* speedups for Prove (similar effects for Verify)
- **No random oracle for proving**
  - ⇒ more efficient composition with other cryptographic proofs
- **Fully associative proofs**
- **Also:**

$$\pi_{I \cup J} = \pi_I + \pi_J$$



*nor verification time—more in a second.

Main Result of This Work

scalar ops in $\mathbb{F}$

First SVC with $\Theta(|J|)$ proving time.

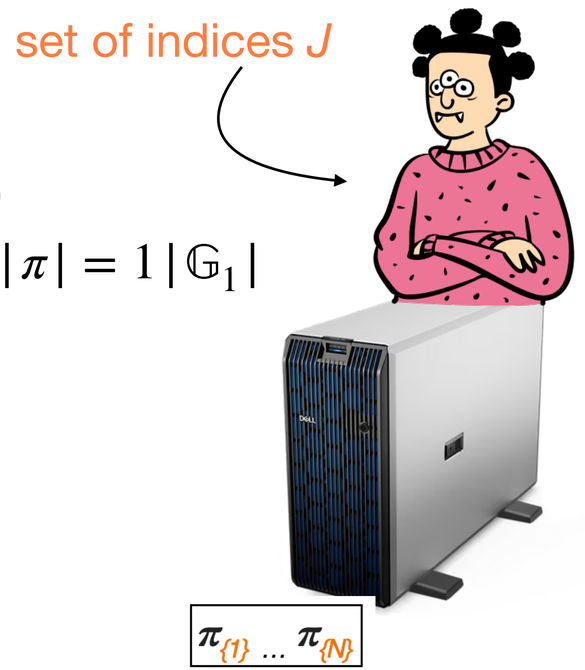
(this is without sacrificing bandwidth*: it has O(1) commitment and proof size)

$|\mathsf{cm}| = |\pi| = 1\,|\mathbb{G}_1|$

"Cute asymptotics you got there—but why would I care?"

- **Concretely faster**
  - 2—3 *orders-of-magnitude* speedups for Prove (similar effects for Verify)
- **No random oracle for proving**
  - ⇒ more efficient composition with other cryptographic proofs
- **Fully associative proofs**
- **Also:**
  - Immediately deployable ("usual universal setup")

$$\pi_{I \cup J} = \pi_I + \pi_J$$



*nor verification time—more in a second.

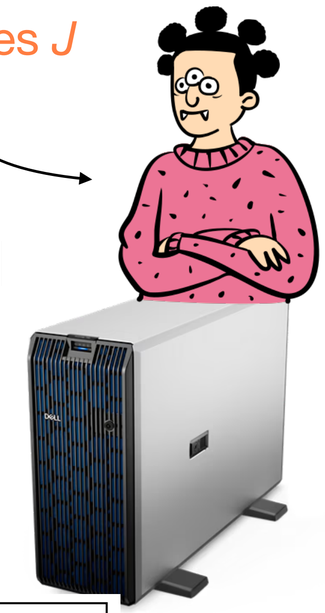
Main Result of This Work

scalar ops in $\mathbb{F}$

First SVC with $\Theta(|J|)$ proving time.

(this is without sacrificing bandwidth*: it has O(1) commitment and proof size)

$|\mathsf{cm}| = |\pi| = 1\,|\mathbb{G}_1|$



$\mathsf{Prove}(J) \rightarrow \pi_J$

F $\in \mathbb{F}^N$

*nor verification time—more in a second.

"Cute asymptotics you got there—but why would I care?"

- **Concretely faster**
  - 2—3 *orders-of-magnitude* speedups for Prove (similar effects for Verify)
- **No random oracle for proving**
  - ⇒ more efficient composition with other cryptographic proofs
- **Fully associative proofs**

$$\pi_{I \cup J} = \pi_I + \pi_J$$

- **Also:**
  - Immediately deployable ("usual universal setup")
  - Aggregatable proofs across commitments and updatable.

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)
This work	$O(\ell)$	1 |G|	reusable; can use deployed ones

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)
This work	$O(\ell)$	1 |G|	reusable; can use deployed ones
aSVC [TAB+20]	$O(\ell \log^2 \ell)$	1 |G|	reusable; can use deployed ones

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)
This work	$O(\ell)$	1 |G|	reusable; can use deployed ones
aSVC [TAB+20]	$O(\ell \log^2 \ell)$	1 |G|	reusable; can use deployed ones

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)
This work	$O(\ell)$	1 |G|	reusable; can use deployed ones
aSVC [TAB+20]	$O(\ell \log^2 \ell)$	1 |G|	reusable; can use deployed ones

- Uses interpolation for aggregation.

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)
This work	$O(\ell)$	1 |G|	reusable; can use deployed ones
aSVC [TAB+20]	$O(\ell \log^2 \ell)$	1 |G|	reusable; can use deployed ones

- Uses interpolation for aggregation.
- Prover 60x-4000x slower (depending on subvector)

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)
This work	$O(\ell)$	1 |G|	reusable; can use deployed ones
aSVC [TAB+20]	$O(\ell \log^2 \ell)$	1 |G|	reusable; can use deployed ones

- Uses interpolation for aggregation.
- Prover 60x-4000x slower (depending on subvector)
- Verifier up to 170x slower

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)
This work	$O(\ell)$	1 |G|	reusable; can use deployed ones
aSVC [TAB+20]	$O(\ell \log^2 \ell)$	1 |G|	reusable; can use deployed ones

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)
This work	$O(\ell)$	1 |G|	reusable; can use deployed ones
aSVC [TAB+20]	$O(\ell \log^2 \ell)$	1 |G|	reusable; can use deployed ones
Pointproofs [GRWZ20]		1 |G|	bespoke larger setup; not deployed

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)
This work	$O(\ell)$	1 |G|	reusable; can use deployed ones
aSVC [TAB+20]	$O(\ell \log^2 \ell)$	1 |G|	reusable; can use deployed ones
Pointproofs [GRWZ20]		1 |G|	bespoke larger setup; not deployed
Linear-Map VC [**C**NR+22]		3 |G|	reusable; can use deployed ones

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)
This work	$O(\ell)$	1 |G|	reusable; can use deployed ones
aSVC [TAB+20]	$O(\ell \log^2 \ell)$	1 |G|	reusable; can use deployed ones
Pointproofs [GRWZ20]	$O\left(\frac{\lambda\ell}{\log(\lambda\ell)}\right)$	1 |G|	bespoke larger setup; not deployed
Linear-Map VC [**C**NR+22]	$O\left(\frac{\lambda\ell}{\log(\lambda\ell)}\right)$	3 |G|	reusable; can use deployed ones

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)
This work	$O(\ell)$	1 |G|	reusable; can use deployed ones
aSVC [TAB+20]	$O(\ell \log^2 \ell)$	1 |G|	reusable; can use deployed ones
Pointproofs [GRWZ20]	$O\left(\frac{\lambda\ell}{\log(\lambda\ell)}\right)$	1 |G|	bespoke larger setup; not deployed
Linear-Map VC [**C**NR+22]	$O\left(\frac{\lambda\ell}{\log(\lambda\ell)}\right)$	3 |G|	reusable; can use deployed ones

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)
This work	$O(\ell)$	1 |G|	reusable; can use deployed ones
aSVC [TAB+20]	$O(\ell \log^2 \ell)$	1 |G|	reusable; can use deployed ones
Pointproofs [GRWZ20]	$O\left(\frac{\lambda\ell}{\log(\lambda\ell)}\right)$	1 |G|	bespoke larger setup; not deployed
Linear-Map VC [**C**NR+22]	$O\left(\frac{\lambda\ell}{\log(\lambda\ell)}\right)$	3 |G|	reusable; can use deployed ones

Aggregation trough random oracle and multi-scalar multiplication.

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)
This work	$O(\ell)$	1 |G|	reusable; can use deployed ones
aSVC [TAB+20]	$O(\ell \log^2 \ell)$	1 |G|	reusable; can use deployed ones
Pointproofs [GRWZ20]	$O\left(\frac{\lambda\ell}{\log(\lambda\ell)}\right)$	1 |G|	bespoke larger setup; not deployed
Linear-Map VC [**C**NR+22]	$O\left(\frac{\lambda\ell}{\log(\lambda\ell)}\right)$	3 |G|	reusable; can use deployed ones

Aggregation trough random oracle and multi-scalar multiplication.

$$\pi_J \leftarrow \sum_{j \in J} H(\ldots)\pi_j$$

Putting This More in Context

Comparison with prior work (this slide: focus on practical schemes)

ℓ := subvector size

Scheme	Proving Time	|proof|	Setup (parameters)
This work	$O(\ell)$	1 |G|	reusable; can use deployed ones
aSVC [TAB+20]	$O(\ell \log^2 \ell)$	1 |G|	reusable; can use deployed ones
Pointproofs [GRWZ20]	$O\left(\frac{\lambda\ell}{\log(\lambda\ell)}\right)$	1 |G|	bespoke larger setup; not deployed
Linear-Map VC [**C**NR+22]	$O\left(\frac{\lambda\ell}{\log(\lambda\ell)}\right)$	3 |G|	reusable; can use deployed ones

Aggregation trough random oracle and multi-scalar multiplication.

$$\pi_J \leftarrow \sum_{j \in J} H(\ldots)\pi_j$$

Asymptotics are due to Pippenger's algorithm. Concretely at least ≈40x slower

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification,
  enforce a polynomial equation

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification,
  enforce a polynomial equation
  - the proof consists of additional
    polynomials to check the equation

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

"Usual" approach [KZG10,TAB+20]:
Polynomial remainder theorem

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(t) = y \iff \exists q(X) \ : \ f(X) - y = q(X)(X - t)$$

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(t) = y \iff \exists q(X) \ : \ f(X) - y = q(X)(X - t)$$

Prover claims *i*-th position is a_i.

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(t) = y \iff \exists q(X) \ : \ f(X) - y = q(X)(X - t)$$

Prover claims *i*-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(t) = y \iff \exists q(X) \ : \ f(X) - y = q(X)(X - t)$$

Prover claims *i*-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(t) = y \iff \exists q(X) \ : \ f(X) - y = q(X)(X - t)$$

Prover claims *i*-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(t) = y \iff \exists q(X) \ : \ f(X) - y = q(X)(X - t)$$

Prover claims *i*-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(t) = y \iff \exists q(X) \ : \ f(X) - y = q(X)(X - t)$$

Prover claims *i*-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(t) = y \iff \exists q(X) \ : \ f(X) - y = q(X)(X - t)$$

Prover claims *i*-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(t) = y \iff \exists q(X) \ : \ f(X) - y = q(X)(X - t)$$

Prover claims i-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(t) = y \iff \exists q(X) \ : \ f(X) - y = q(X)(X - t)$$

Prover claims i-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(t) = y \iff \exists q(X) \ : \ f(X) - y = q(X)(X - t)$$

Prover claims i-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Issue: proving ℓ positions → interpolation → $O(\ell \log^2 \ell)$ time

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

A different approach:

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(t) = y \iff \exists q(X) \ : \ f(X) - y = q(X)(X - t)$$

Prover claims i-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Issue: proving ℓ positions → interpolation → $O(\ell \log^2 \ell)$ time

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

A different approach:

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(i) = y \iff \exists q(X) \;.\; f(X) - y = q(X)(X - i)$$

Prover claims i-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Issue: proving ℓ positions → interpolation → $O(\ell \log^2 \ell)$ time

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

A different approach:

$$f_{\mathbf{a}}(X)\lambda_j(X)$$

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(t) = y \iff \exists q(X) \;.\; f(X) - y = q(X)(X - t)$$

Prover claims i-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Issue: proving ℓ positions → interpolation → $O(\ell log^2 \ell)$ time

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

A different approach:

$$f_{\mathbf{a}}(X)\lambda_j(X) \equiv \qquad \pmod{t(X)}$$

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(i) = y \iff \exists q(X) \;.\; f(X) - y = q(X)(X - i)$$

Prover claims i-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Issue: proving ℓ positions → interpolation → $O(\ell \log^2 \ell)$ time

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

A different approach:

$$f_{\mathbf{a}}(X)\lambda_j(X) \equiv \qquad \pmod{t(X)}$$

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(i) = y \iff \exists q(X) \;.\; f(X) - y = q(X)(X - i)$$

Prover claims i-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Issue: proving ℓ positions → interpolation → $O(\ell log^2 \ell)$ time

t(X): vanishing polynomial in {1,…,N}

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

A different approach:

$$f_{\mathbf{a}}(X)\lambda_j(X) \equiv a_j\lambda_j(X) \pmod{t(X)}$$

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(i) = y \iff \exists q(X) \;.\; f(X) - y = q(X)(X - i)$$

Prover claims i-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Issue: proving ℓ positions → interpolation → $O(\ell \log^2 \ell)$ time

t(X): vanishing polynomial in {1,…,N}

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

A different approach:

$$f_{\mathbf{a}}(X)\lambda_j(X) \equiv a_j\lambda_j(X) \pmod{t(X)}$$

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(i) = y \iff \exists q(X) \;.\; f(X) - y = q(X)(X - i)$$

Prover claims i-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Issue: proving ℓ positions → interpolation → $O(\ell log^2 \ell)$ time

t(X): vanishing polynomial in {1,…,N}

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

A different approach:

$$f_{\mathbf{a}}(X)\lambda_j(X) \equiv a_j\lambda_j(X) \pmod{t(X)}$$

New equation:

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(i) = y \iff \exists q(X) \;.\; f(X) - y = q(X)(X - i)$$

Prover claims i-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Issue: proving ℓ positions → interpolation → $O(\ell log^2 \ell)$ time

t(X): vanishing polynomial in {1,…,N}

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(i) = y \iff \exists q(X) \;.\; f(X) - y = q(X)(X - i)$$

Prover claims i-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Issue: proving ℓ positions → interpolation → $O(\ell \log^2 \ell)$ time

A different approach:

$$f_{\mathbf{a}}(X)\lambda_j(X) \equiv a_j\lambda_j(X) \pmod{t(X)}$$

New equation:

$$f_{\mathbf{a}}(X)\lambda_j(X) = q'_j(X) \cdot t(X) + a_j\lambda_j(X)$$

t(X): vanishing polynomial in {1,…,N}

The Inner Workings of the SVC in This Work

We want to commit to $\boxed{a_1, \ldots, a_n}$

Common approach:

- encode it as a polynomial
- for proving/verification, enforce a polynomial equation
  - the proof consists of additional polynomials to check the equation

Encoding via interpolation:

$$f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$$

Lagrange polynomials

Intuitively, as a commitment the client gets an encoding of $f_{\mathbf{a}}(X)$.

"Usual" approach [KZG10,TAB+20]:

Polynomial remainder theorem

$$f(t) = y \iff \exists q(X) \;.\; f(X) - y = q(X)(X - t)$$

Prover claims i-th position is a_i.

→ In order to check that we check that $f_{\mathbf{a}}(i) = a_i$

Issue: proving ℓ positions → interpolation → $O(\ell \log^2 \ell)$ time

A different approach:

$$f_{\mathbf{a}}(X)\lambda_j(X) \equiv a_j\lambda_j(X) \pmod{t(X)}$$

New equation:

$$f_{\mathbf{a}}(X)\lambda_j(X) = q'_j(X) \cdot t(X) + a_j\lambda_j(X)$$

t(X): vanishing polynomial in {1,…,N}

Aggregation in the New Scheme

Aggregation in the New Scheme

New equation:

$$f_{\mathbf{a}}(X)\lambda_j(X) = q_j'(X) \cdot t(X) + a_j\lambda_j(X)$$ (to prove index j)

Aggregation in the New Scheme

New equation:

$$f_{\mathbf{a}}(X)\lambda_j(X) = q_j'(X) \cdot t(X) + a_j\lambda_j(X)$$ (to prove index j)

$$f_{\mathbf{a}}(X)\lambda_i(X) = q_i'(X) \cdot t(X) + a_i\lambda_i(X)$$ (to prove index i)

Aggregation in the New Scheme

New equation:

$$f_{\mathbf{a}}(X)\lambda_j(X) = q_j'(X) \cdot t(X) + a_j\lambda_j(X) \quad \text{(to prove index j)}$$

$$+$$

$$f_{\mathbf{a}}(X)\lambda_i(X) = q_i'(X) \cdot t(X) + a_i\lambda_i(X) \quad \text{(to prove index i)}$$

Aggregation in the New Scheme

New equation:

$$f_{\mathbf{a}}(X)\lambda_j(X) = q_j'(X) \cdot t(X) + a_j\lambda_j(X) \quad \text{(to prove index j)}$$

$$+$$

$$f_{\mathbf{a}}(X)\lambda_i(X) = q_i'(X) \cdot t(X) + a_i\lambda_i(X) \quad \text{(to prove index i)}$$

Aggregation in the New Scheme

New equation:

$$f_{\mathbf{a}}(X)\lambda_j(X) = q'_j(X) \cdot t(X) + a_j\lambda_j(X) \quad \text{(to prove index j)}$$

+

$$f_{\mathbf{a}}(X)\lambda_i(X) = q'_i(X) \cdot t(X) + a_i\lambda_i(X) \quad \text{(to prove index i)}$$

↓

$$f_{\mathbf{a}}(X)(\lambda_i(X) + \lambda_i(X)) = (q'_j(X) + q'_i(X)) \cdot t(X) + (a_j\lambda_j(X) + a_i\lambda_i(X)) \quad \text{(to prove indices \{i,j\})}$$

Aggregation in the New Scheme

New equation:

$$f_{\mathbf{a}}(X)\lambda_j(X) = q'_j(X) \cdot t(X) + a_j\lambda_j(X) \quad \text{(to prove index j)}$$

$$+$$

$$f_{\mathbf{a}}(X)\lambda_i(X) = q'_i(X) \cdot t(X) + a_i\lambda_i(X) \quad \text{(to prove index i)}$$

$$f_{\mathbf{a}}(X)(\lambda_i(X) + \lambda_i(X)) = (q'_j(X) + q'_i(X)) \cdot t(X) + (a_j\lambda_j(X) + a_i\lambda_i(X)) \quad \text{(to prove indices \{i,j\})}$$

Aggregation in the New Scheme

New equation:

$$f_{\mathbf{a}}(X)\lambda_j(X) = q'_j(X) \cdot t(X) + a_j\lambda_j(X) \quad \text{(to prove index j)}$$

+

$$f_{\mathbf{a}}(X)\lambda_i(X) = q'_i(X) \cdot t(X) + a_i\lambda_i(X) \quad \text{(to prove index i)}$$

$$f_{\mathbf{a}}(X)(\lambda_i(X) + \lambda_i(X)) = (q'_j(X) + q'_i(X)) \cdot t(X) + (a_j\lambda_j(X) + a_i\lambda_i(X)) \quad \text{(to prove indices {i,j})}$$

$$\pi_{I \cup J} = \pi_I + \pi_J$$

Wrapping Up

Wrapping Up

- Simple ideas can give you a lot of mileage sometimes

Wrapping Up

- Simple ideas can give you a lot of mileage sometimes
- Linear time SVCs can be substantially faster than prior approaches

Wrapping Up

- Simple ideas can give you a lot of mileage sometimes
- Linear time SVCs can be substantially faster than prior approaches
  - (this construction at least is and without sacrificing almost anything)

Wrapping Up

- Simple ideas can give you a lot of mileage sometimes
- Linear time SVCs can be substantially faster than prior approaches
  - (this construction at least is and without sacrificing almost anything)
- *Concrete Application Setting*: faster Verifiable Database proving

Wrapping Up

qedb: Expressive and Modular Verifiable Databases (without SNARKs)

Vincenzo Botta
vincenzo@provably.ai
Provably Technologies
Roma, Italy

Simone Bottoni
simone@provably.ai
Provably Technologies
Varese, Italy

Matteo Campanelli
binarywhalesinternaryseas@gmail.com
Offchain Labs and University of Tartu

Emanuele Ragnoli
emanuele@provably.ai
Provably Technologies
Warsaw, Poland

Alberto Trombetta
alberto@provably.ai
Provably Technologies
Como, Italy

To appear at ACM CCS 2026

- Simple ideas can give you a lot of mileage sometimes
- Linear time SVCs can be substantially faster than prior approaches
  - (this construction at least is and without sacrificing almost anything)
- *Concrete Application Setting*: faster Verifiable Database proving

Wrapping Up

- Simple ideas can give you a lot of mileage sometimes
- Linear time SVCs can be substantially faster than prior approaches
  - (this construction at least is and without sacrificing almost anything)
- *Concrete Application Setting*: faster Verifiable Database proving

Wrapping Up

- Simple ideas can give you a lot of mileage sometimes
- Linear time SVCs can be substantially faster than prior approaches
  - (this construction at least is and without sacrificing almost anything)
- *Concrete Application Setting*: faster Verifiable Database proving

Open questions:

Wrapping Up

- Simple ideas can give you a lot of mileage sometimes
- Linear time SVCs can be substantially faster than prior approaches
  - (this construction at least is and without sacrificing almost anything)
- *Concrete Application Setting*: faster Verifiable Database proving

Open questions:

- Current security for *subvector* opening is in the AGM. Falsifiable assumptions?

Wrapping Up

- Simple ideas can give you a lot of mileage sometimes
- Linear time SVCs can be substantially faster than prior approaches
  - (this construction at least is and without sacrificing almost anything)
- *Concrete Application Setting*: faster Verifiable Database proving

Open questions:

- Current security for *subvector* opening is in the AGM. Falsifiable assumptions?
- SVC with O(|subvector|) proving are an example of a sublinear-time scheme (sublinear in N, the *whole* vector's size)

Wrapping Up

- Simple ideas can give you a lot of mileage sometimes
- Linear time SVCs can be substantially faster than prior approaches
  - (this construction at least is and without sacrificing almost anything)
- *Concrete Application Setting*: faster Verifiable Database proving

Open questions:

- Current security for *subvector* opening is in the AGM. Falsifiable assumptions?
- SVC with O(|subvector|) proving are an example of a sublinear-time scheme (sublinear in N, the *whole* vector's size)
  - Subvectors are "easy" as such (This work shows how much to push that sublinearity, but sublinearity was already there)

Wrapping Up

- Simple ideas can give you a lot of mileage sometimes
- Linear time SVCs can be substantially faster than prior approaches
  - (this construction at least is and without sacrificing almost anything)
- *Concrete Application Setting*: faster Verifiable Database proving

Open questions:

- Current security for *subvector* opening is in the AGM. Falsifiable assumptions?
- SVC with O(|subvector|) proving are an example of a sublinear-time scheme (sublinear in N, the *whole* vector's size)
  - Subvectors are "easy" as such (This work shows how much to push that sublinearity, but sublinearity was already there)
  - Can we find other practical sublinear proving for "harder" settings?

Wrapping Up

On the Power of Polynomial Preprocessing:
Proving Computations in Sublinear Time, and More

Matteo Campanelli[1], Mario Carrillo[2], Ignacio Cascudo[3], Dario Fiore[3], Danilo Francati[4⋆], and Rosario Gennaro[5]

- Simple ideas can give you a lot of mileage sometimes
- Linear time SVCs can be substantially faster than prior approaches
  - (this construction at least is and without sacrificing almost anything)
- *Concrete Application Setting*: faster Verifiable Database proving

Open questions:

- Current security for *subvector* opening is in the AGM. Falsifiable assumptions?
- SVC with O(|subvector|) proving are an example of a sublinear-time scheme (sublinear in N, the *whole* vector's size)
  - Subvectors are "easy" as such (This work shows how much to push that sublinearity, but sublinearity was already there)
  - Can we find other practical sublinear proving for "harder" settings?

Wrapping Up

On the Power of Polynomial Preprocessing:
Proving Computations in Sublinear Time, and More

Matteo Campanelli[1], Mario Carrillo[2], Ignacio Cascudo[3], Dario Fiore[3], Danilo Francati[4⋆], and Rosario Gennaro[5]

- Simple ideas can give you a lot of mileage sometimes
- Linear time SVCs can be substantially faster than prior approaches
  - (this construction at least is and without sacrificing almost anything)
- *Concrete Application Setting*: faster Verifiable Database proving

Open questions:

- Current security for *subvector* opening is in the AGM. Falsifiable assumptions?
- SVC with O(|subvector|) proving are an example of a sublinear-time scheme (sublinear in N, the *whole* vector's size)
  - Subvectors are "easy" as such (This work shows how much to push that sublinearity, but sublinearity was already there)
  - Can we find other practical sublinear proving for "harder" settings?
- *Associativity:* How can we study when associativity of proofs is even possible in principle?

Wrapping Up

On the Power of Polynomial Preprocessing:
Proving Computations in Sublinear Time, and More

Matteo Campanelli[1], Mario Carrillo[2], Ignacio Cascudo[3], Dario Fiore[3], Danilo Francati[4⋆], and Rosario Gennaro[5]

- Simple ideas can give you a lot of mileage sometimes
- Linear time SVCs can be substantially faster than prior approaches
  - (this construction at least is and without sacrificing almost anything)
- *Concrete Application Setting*: faster Verifiable Database proving

Open questions:

- Current security for *subvector* opening is in the AGM. Falsifiable assumptions?
- SVC with O(|subvector|) proving are an example of a sublinear-time scheme (sublinear in N, the *whole* vector's size)
  - Subvectors are "easy" as such (This work shows how much to push that sublinearity, but sublinearity was already there)
  - Can we find other practical sublinear proving for "harder" settings?
- *Associativity:* How can we study when associativity of proofs is even possible in principle?
- Can we build O(|subvector|) proving and associative proofs in SVC from lattices?

Wrapping Up

On the Power of Polynomial Preprocessing:
Proving Computations in Sublinear Time, and More

Matteo Campanelli[1], Mario Carrillo[2], Ignacio Cascudo[3], Dario Fiore[3], Danilo Francati[4⋆], and Rosario Gennaro[5]

- Simple ideas can give you a lot of mileage sometimes
- Linear time SVCs can be substantially faster than prior approaches
  - (this construction at least is and without sacrificing almost anything)
- *Concrete Application Setting*: faster Verifiable Database proving

Open questions:

Thank you! Questions?

- Current security for *subvector* opening is in the AGM. Falsifiable assumptions?
- SVC with O(|subvector|) proving are an example of a sublinear-time scheme (sublinear in N, the *whole* vector's size)
  - Subvectors are "easy" as such (This work shows how much to push that sublinearity, but sublinearity was already there)
  - Can we find other practical sublinear proving for "harder" settings?
- *Associativity:* How can we study when associativity of proofs is even possible in principle?
- Can we build O(|subvector|) proving and associative proofs in SVC from lattices?