Side-channel protections for Picnic signatures

CHES 2021

Diego F. Aranha[1] Sebastian Berndt[2] Thomas Eisenbarth[2] Okan Seker[2]
Akira Takahashi[1] Luca Wilke[2] Greg Zaverucha[3]

[1]Aarhus University, Denmark

[2]University of Lübeck, Germany

[3]Microsoft Research, USA

Background & Motivation: NIST PQC Standardization Round 3

Finalists

- CRYSTALS-DILITHIUM
- Falcon
- Rainbow

Alternates

- GeMSS
- Picnic
- SPHINCS+

- Side-channel resilience is becoming more relevant
- Little study on side-channel resilience of **Picnic**/MPC-in-the-head paradigm

Picnic & side-channel security

- Fiat–Shamir-type signature from **MPC-in-the-head** ZKP [IKOS07]

☺ No number-theoretic assumptions
  - Block cipher
  - Hash function (modeled as RO)

☺ Various parameters

Picnic1

☹ Known to be vulnerable to DPA [GSE20]

☹ Existing countermeasure breaks interoperability with verification [SBWE20]

☹ Also increases signature size

Picnic3

- Follows MPC-in-the-head with **preprocessing** paradigm [KKW18]
- More compact signature
- No side-channel evaluation yet

Picnic & side-channel security

- Fiat–Shamir-type signature from **MPC-in-the-head** ZKP [IKOS07]

☺ No number-theoretic assumptions
  - Block cipher
  - Hash function (modeled as RO)

☺ Various parameters

Picnic1

☹ Known to be vulnerable to DPA [GSE20]

☹ Existing countermeasure breaks interoperability with verification [SBWE20]

☹ Also increases signature size

Picnic3

- Follows MPC-in-the-head with **preprocessing** paradigm [KKW18]
- More compact signature
- No side-channel evaluation yet

Picnic & side-channel security

- Fiat–Shamir-type signature from **MPC-in-the-head** ZKP [IKOS07]

☺ No number-theoretic assumptions
  - Block cipher
  - Hash function (modeled as RO)

☺ Various parameters

Picnic1

☹ Known to be vulnerable to DPA [GSE20]

☹ Existing countermeasure breaks interoperability with verification [SBWE20]

☹ Also increases signature size

Picnic3

- Follows MPC-in-the-head with **preprocessing** paradigm [KKW18]
- More compact signature
- No side-channel evaluation yet

Our goal

- Side-channel evaluation of Picnic3 / MPC-in-the-head with preprocessing
- Maintain **interoperability** and **signature size** while applying masking countermeasures

This work

- Side-channel vulnerabilities of unprotected **Picnic3**
  - Attack I extends [GSE20]
  - Attack II is new
- Generic approach to mask ZKP using MPCitH with **preprocessing**
  - Proof for t-probing security
  - Supported by **`maskVerif`** formal verification tool [BBC+19]
  - Possible to trade-off **provable security** for **lower masking overhead**
- First-order masked implementation of **Picnic3** & **SHA-3**
- Practical electromagnetic (EM) leakage analysis

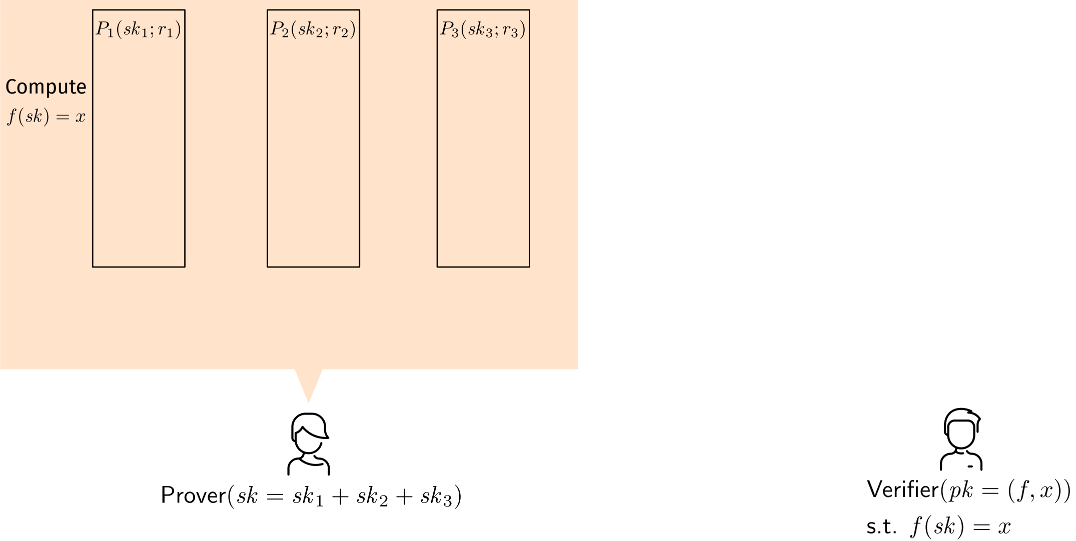
Side-channel Attacks on Picnic3

Zero-knowledge proof using the MPC-in-the-head [IKOS07, GMO16]

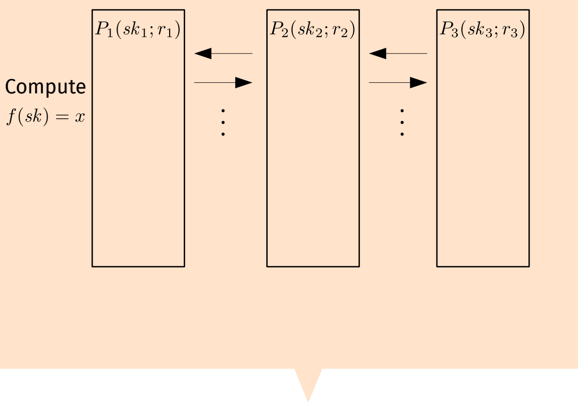
Prover$(sk = sk_1 + sk_2 + sk_3)$

Verifier$(pk = (f, x))$
s.t. $f(sk) = x$

Zero-knowledge proof using the MPC-in-the-head [IKOS07, GMO16]



Zero-knowledge proof using the MPC-in-the-head [IKOS07, GMO16]

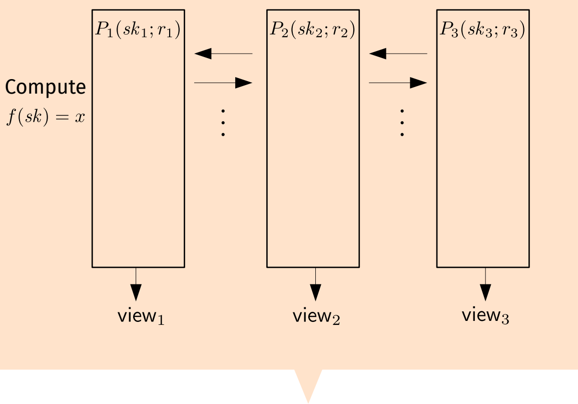


Prover$(sk = sk_1 + sk_2 + sk_3)$

Verifier$(pk = (f, x))$

s.t. $f(sk) = x$

Zero-knowledge proof using the MPC-in-the-head [IKOS07, GMO16]

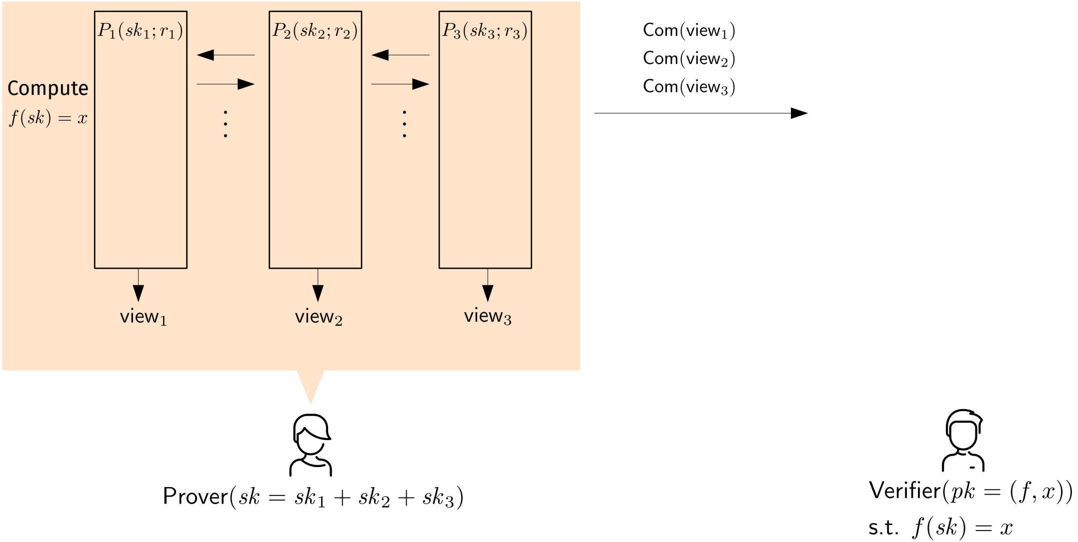


Prover$(sk = sk_1 + sk_2 + sk_3)$

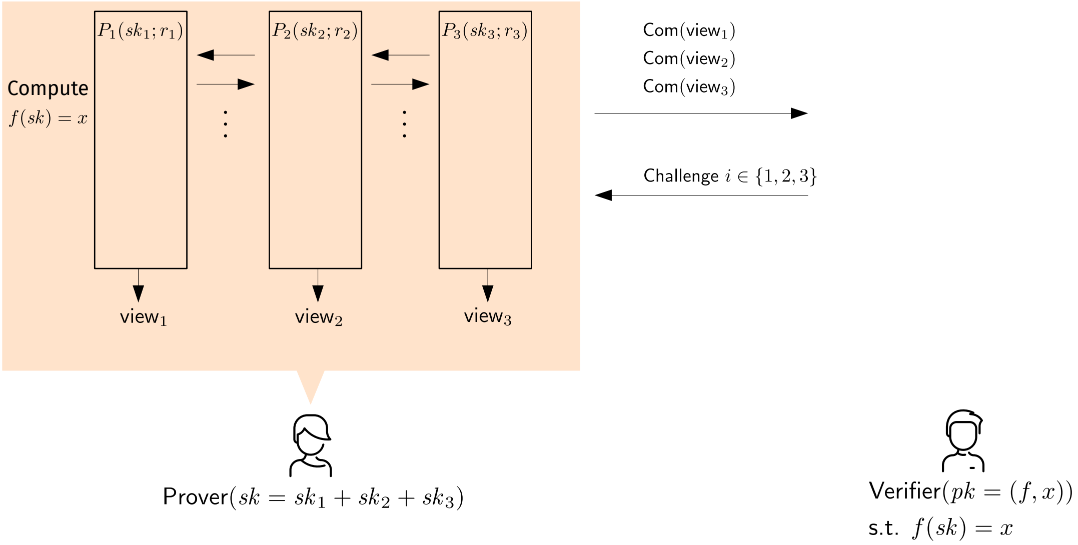
Verifier$(pk = (f, x))$

s.t. $f(sk) = x$

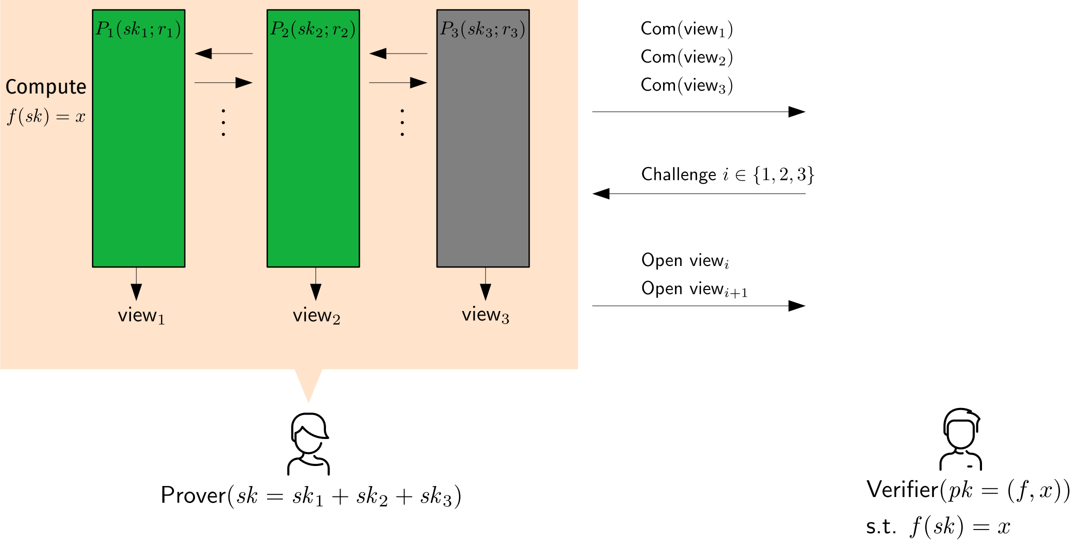
Zero-knowledge proof using the MPC-in-the-head [IKOS07, GMO16]



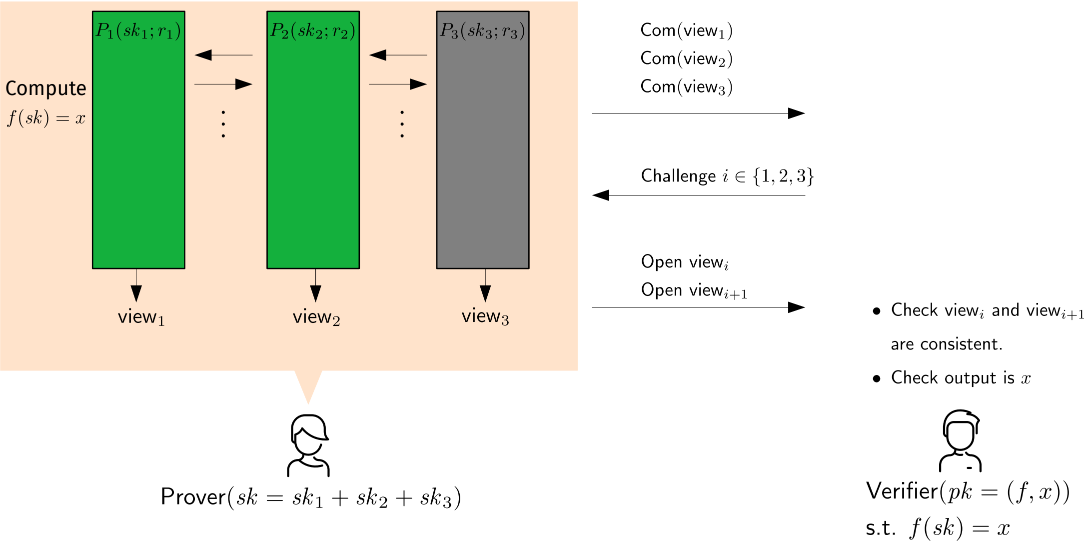
Zero-knowledge proof using the MPC-in-the-head [IKOS07, GMO16]



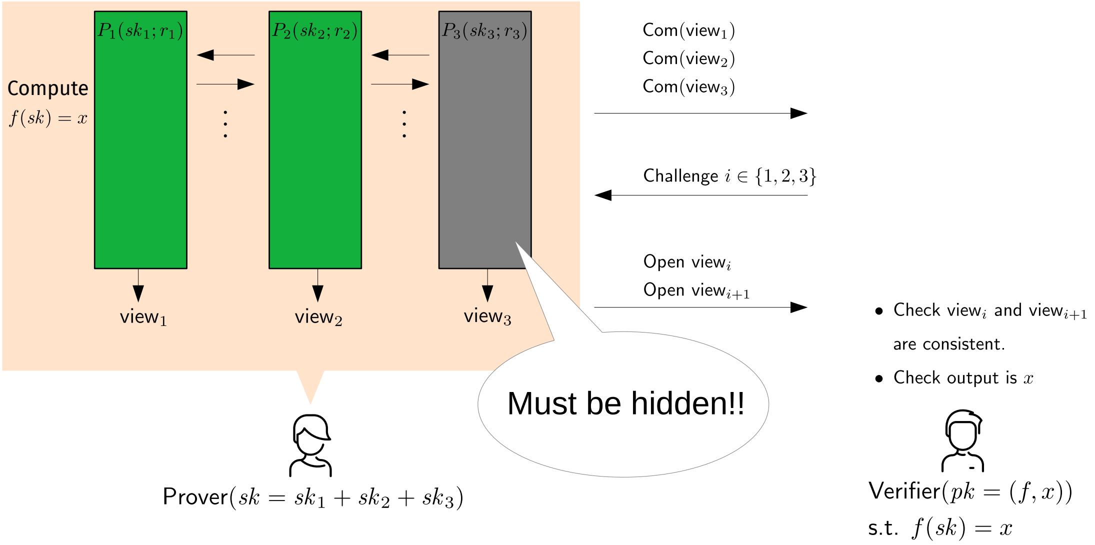
Zero-knowledge proof using the MPC-in-the-head [IKOS07, GMO16]



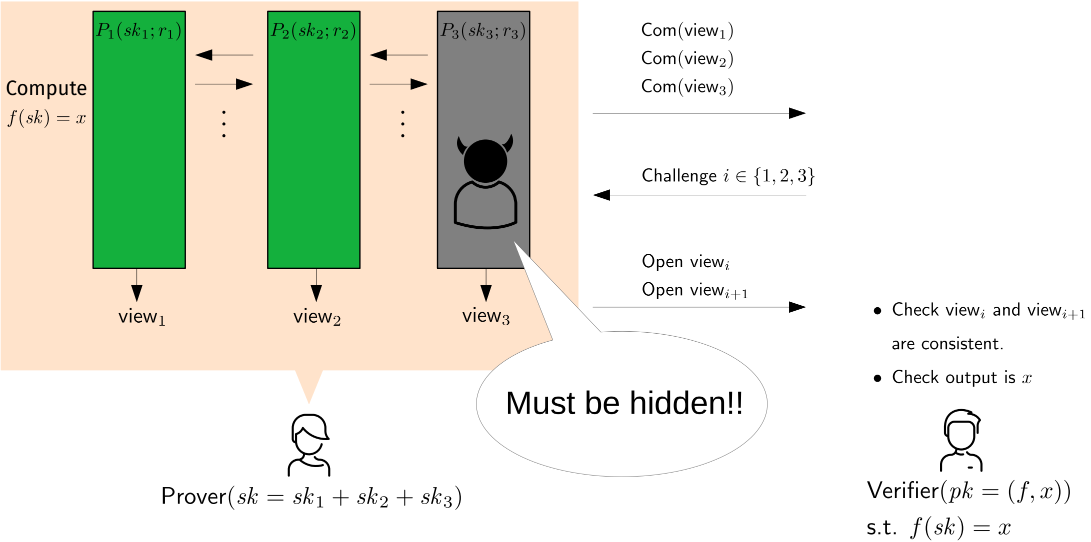
Zero-knowledge proof using the MPC-in-the-head [IKOS07, GMO16]



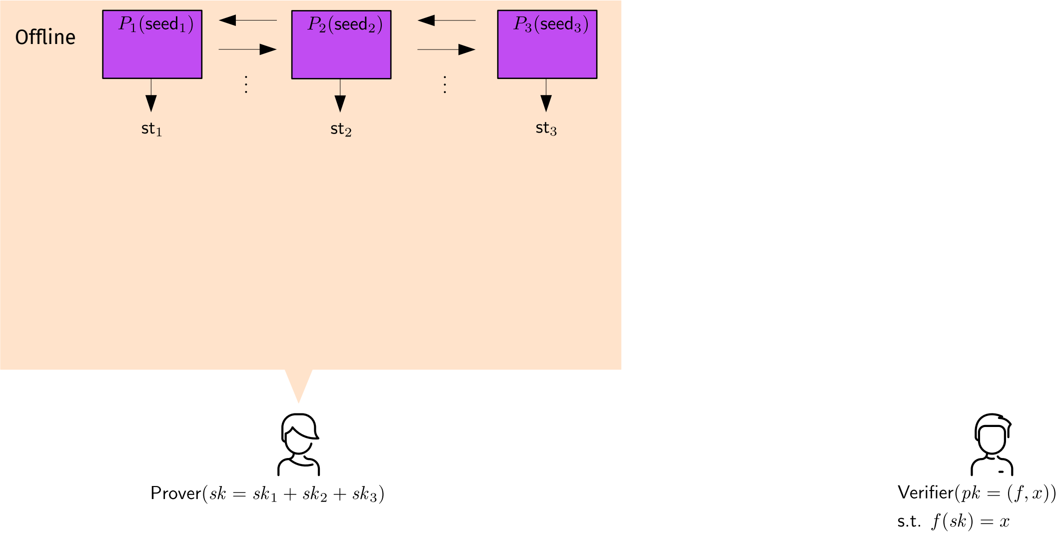
Attack I: Probing the unopened party (extending [GSE20])



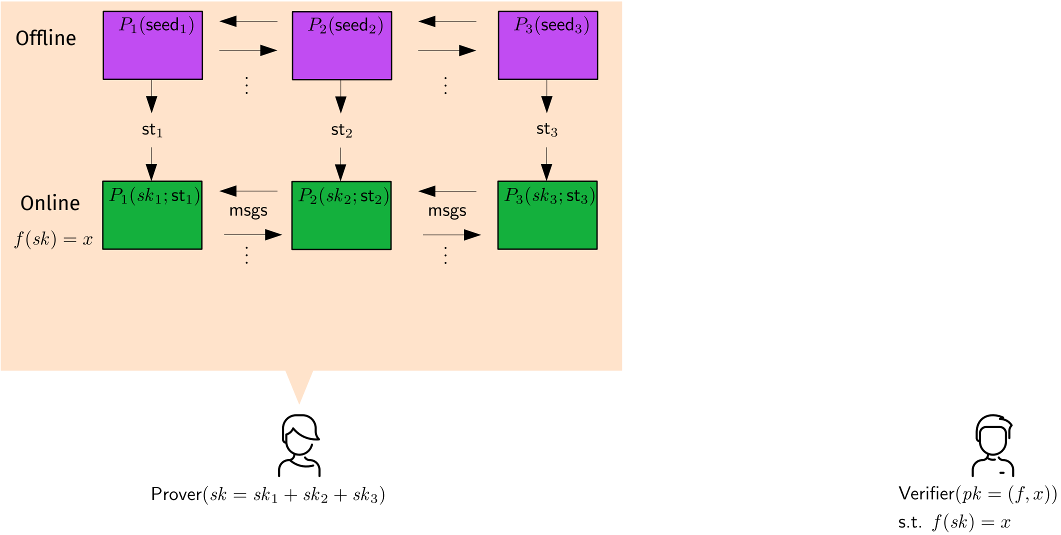
Attack I: Probing the unopened party (extending [GSE20])



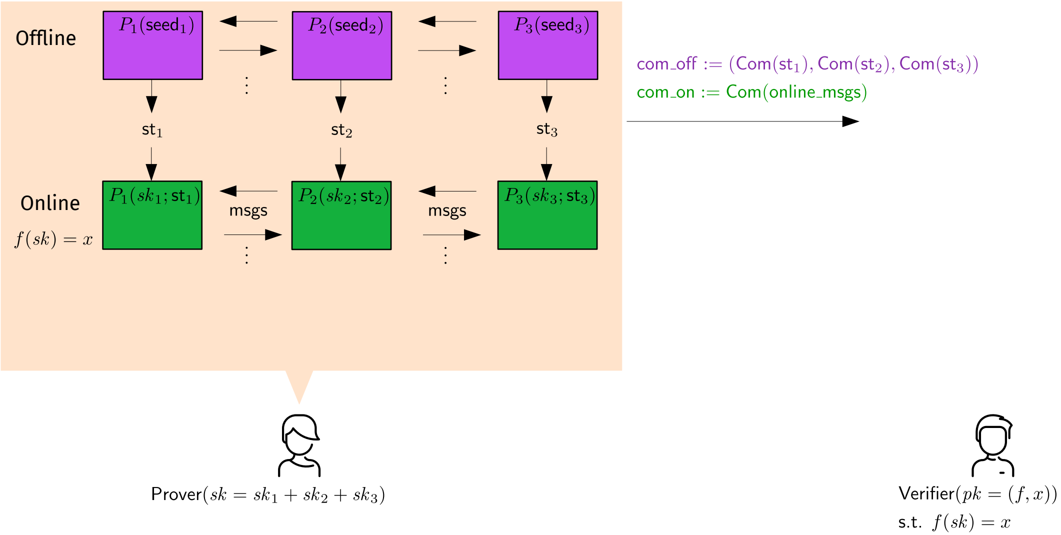
Picnic3: MPC-in-the-head with preprocessing [KKW18, KZ20]



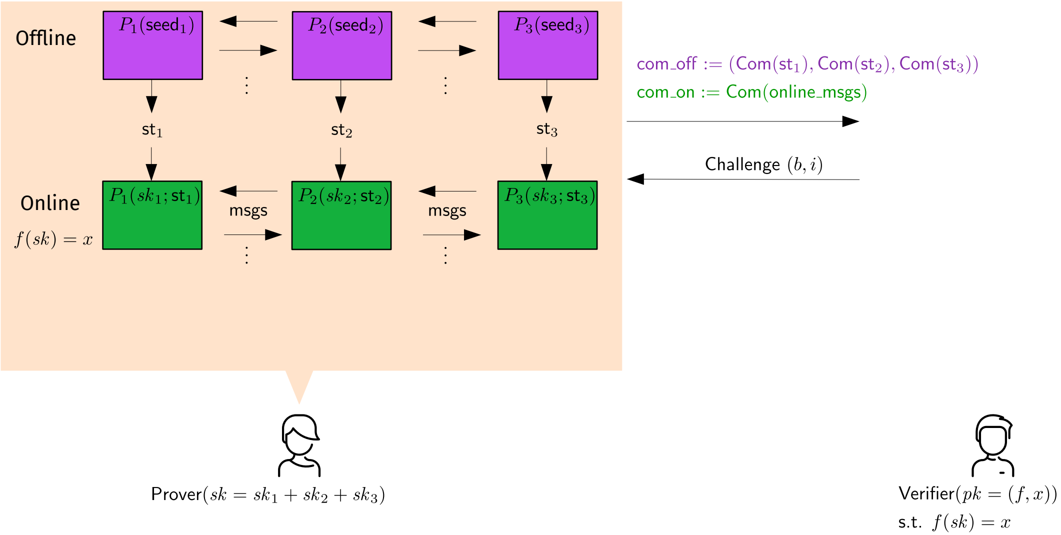
Picnic3: MPC-in-the-head with preprocessing [KKW18, KZ20]



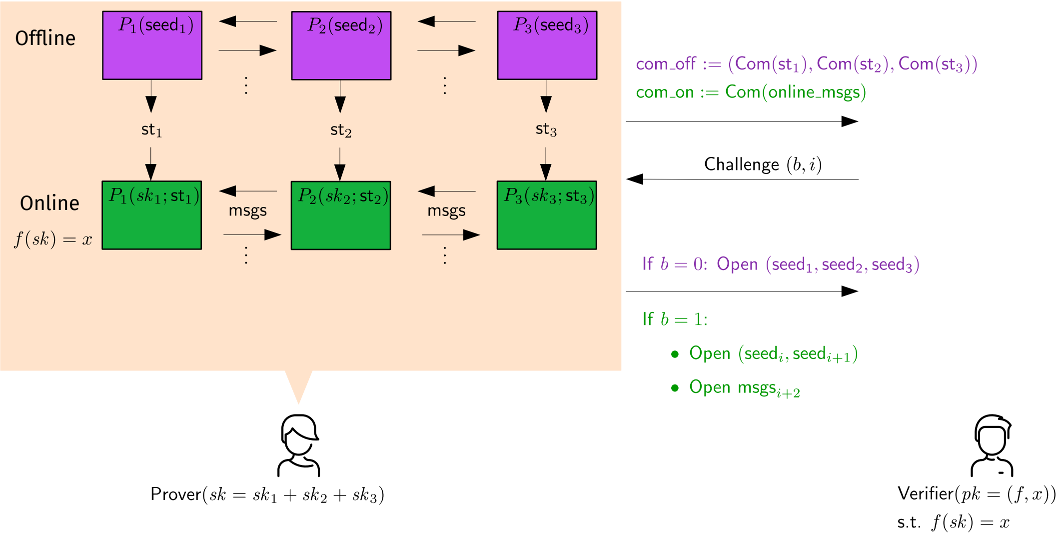
Picnic3: MPC-in-the-head with preprocessing [KKW18, KZ20]



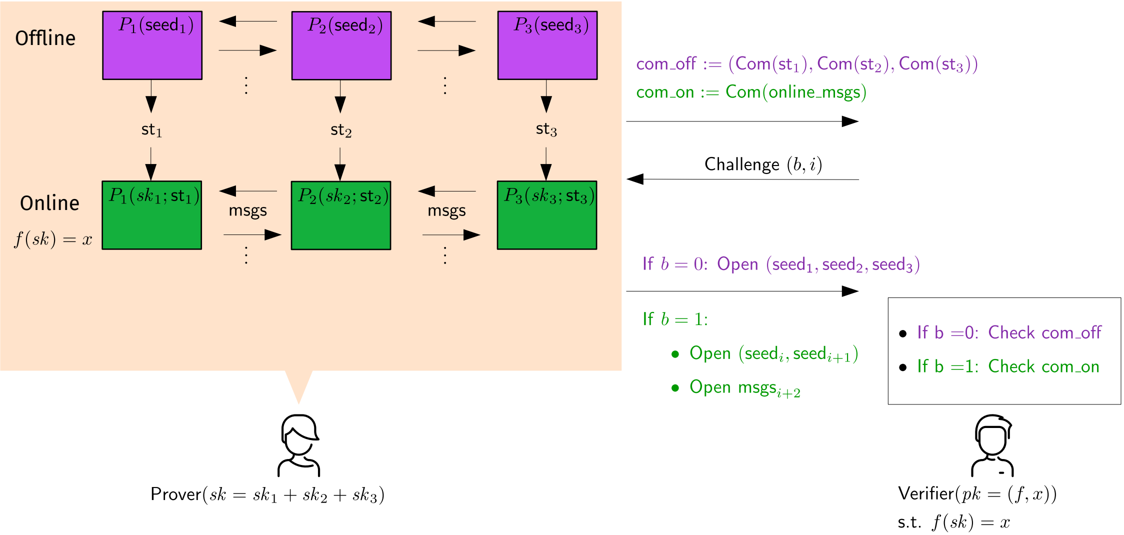
Picnic3: MPC-in-the-head with preprocessing [KKW18, KZ20]



Picnic3: MPC-in-the-head with preprocessing [KKW18, KZ20]



Picnic3: MPC-in-the-head with preprocessing [KKW18, KZ20]



Why preprocessing? – Multiplication with Beaver triples

Inputs $:[x] = (x_1, \ldots, x_N)$ and $[y] = (y_1, \ldots, y_N)$

Output $:[z] = (z_1, \ldots, z_N)$ such that $z = xy$

Offline

- Generate many random triples $([\lambda^x], [\lambda^y], [\lambda^{xy}])$ with $\lambda^{xy} = \lambda^x \lambda^y$
- Easy in MPCitH:

$$\lambda_N^{xy} := \left(\sum_{i=1}^{N} \lambda_i^x\right)\left(\sum_{i=1}^{N} \lambda_i^y\right) - \sum_{i=1}^{N-1} \lambda_i^{xy}$$

Online

- Observation: $xy = ((x + \lambda^x) - \lambda^x)((y + \lambda^y) - \lambda^y)$
- Reconstruct $\hat{x} := x + \lambda^x$ and $\hat{y} := y + \lambda^y$
- Compute

$$[z] = \hat{x}\hat{y} - \hat{x}[\lambda^y] - \hat{y}[\lambda^x] - [\lambda^{xy}]$$

No non-linear operations in the online phase!

WARNING: New attack surface arises..

Why preprocessing? – Multiplication with Beaver triples

Inputs $:[x] = (x_1, \dots, x_N)$ and $[y] = (y_1, \dots, y_N)$

Output $:[z] = (z_1, \dots, z_N)$ such that $z = xy$

Offline

- Generate many random triples $([\lambda^x], [\lambda^y], [\lambda^{xy}])$ with $\lambda^{xy} = \lambda^x \lambda^y$
- Easy in MPCitH:

$$\lambda_N^{xy} := \left(\sum_{i=1}^{N} \lambda_i^x\right)\left(\sum_{i=1}^{N} \lambda_i^y\right) - \sum_{i=1}^{N-1} \lambda_i^{xy}$$

Online

- Observation:
  $xy = ((x + \lambda^x) - \lambda^x)((y + \lambda^y) - \lambda^y)$
- Reconstruct $\hat{x} := x + \lambda^x$ and $\hat{y} := y + \lambda^y$
- Compute

$$[z] = \hat{x}\hat{y} - \hat{x}[\lambda^y] - \hat{y}[\lambda^x] - [\lambda^{xy}]$$

No non-linear operations in the online phase!

WARNING: New attack surface arises..

Why preprocessing? – Multiplication with Beaver triples

Inputs $:[x] = (x_1, \dots, x_N)$ and $[y] = (y_1, \dots, y_N)$

Output $:[z] = (z_1, \dots, z_N)$ such that $z = xy$

Offline

- Generate many random triples $([\lambda^x], [\lambda^y], [\lambda^{xy}])$ with $\lambda^{xy} = \lambda^x \lambda^y$
- Easy in MPCitH:

$$\lambda_N^{xy} := \left(\sum_{i=1}^{N} \lambda_i^x\right)\left(\sum_{i=1}^{N} \lambda_i^y\right) - \sum_{i=1}^{N-1} \lambda_i^{xy}$$

Online

- Observation:
  $xy = ((x + \lambda^x) - \lambda^x)((y + \lambda^y) - \lambda^y)$
- Reconstruct $\hat{x} := x + \lambda^x$ and $\hat{y} := y + \lambda^y$
- Compute

$$[z] = \hat{x}\hat{y} - \hat{x}[\lambda^y] - \hat{y}[\lambda^x] - [\lambda^{xy}]$$

No non-linear operations in the online phase!

WARNING: New attack surface arises..

Why preprocessing? – Multiplication with Beaver triples

Inputs $:[x] = (x_1, \dots, x_N)$ and $[y] = (y_1, \dots, y_N)$

Output $:[z] = (z_1, \dots, z_N)$ such that $z = xy$

Offline

- Generate many random triples $([\lambda^x], [\lambda^y], [\lambda^{xy}])$ with $\lambda^{xy} = \lambda^x \lambda^y$
- Easy in MPCitH:

$$\lambda_N^{xy} := \left(\sum_{i=1}^{N} \lambda_i^x\right)\left(\sum_{i=1}^{N} \lambda_i^y\right) - \sum_{i=1}^{N-1} \lambda_i^{xy}$$

Online

- Observation:

$$xy = ((x + \lambda^x) - \lambda^x)((y + \lambda^y) - \lambda^y)$$

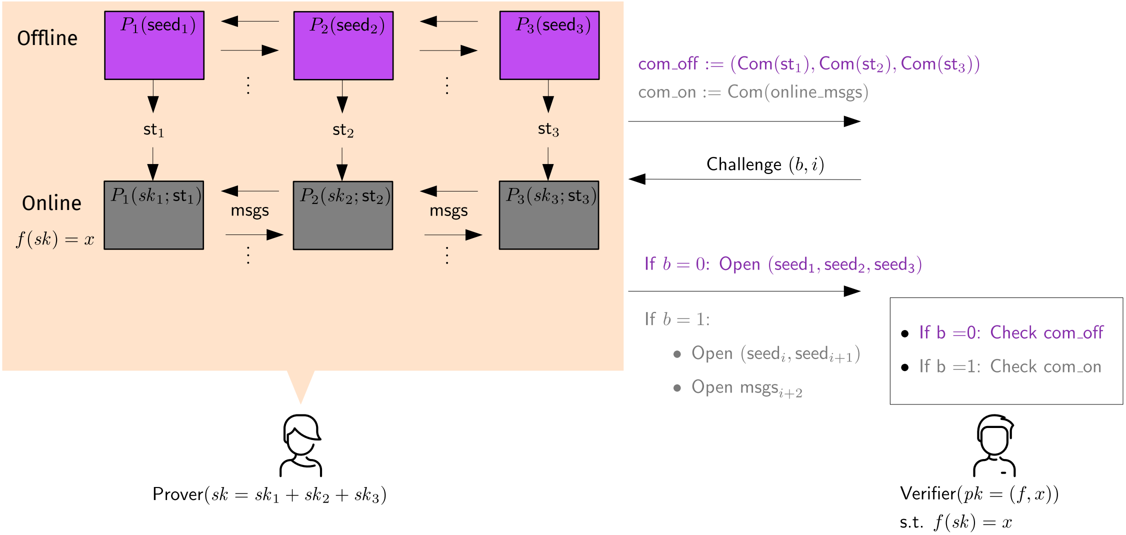
- Reconstruct $\hat{x} := x + \lambda^x$ and $\hat{y} := y + \lambda^y$
- Compute

$$[z] = \hat{x}\hat{y} - \hat{x}[\lambda^y] - \hat{y}[\lambda^x] - [\lambda^{xy}]$$

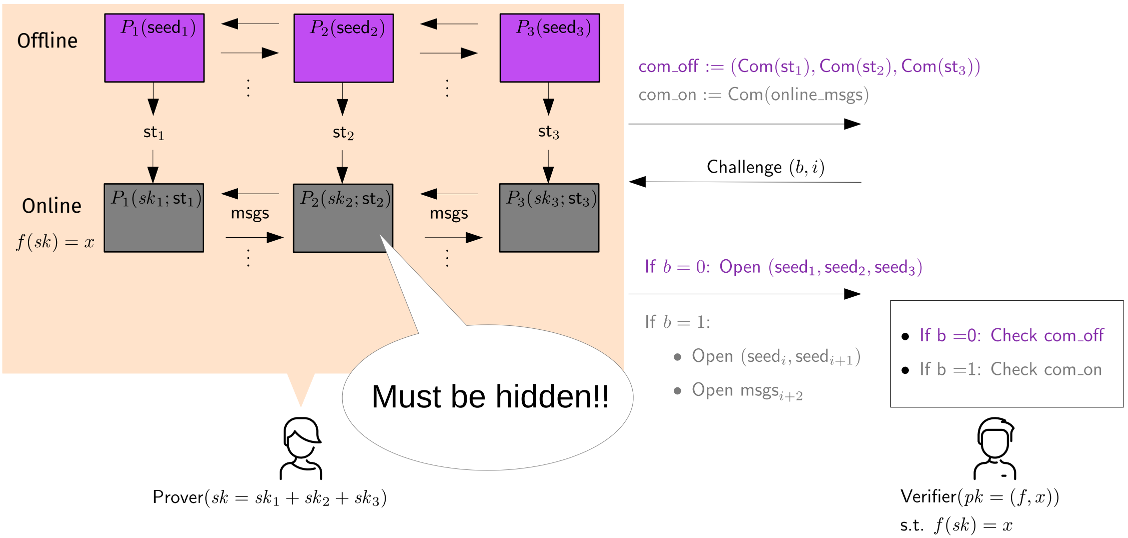
No non-linear operations in the online phase!

WARNING: New attack surface arises..

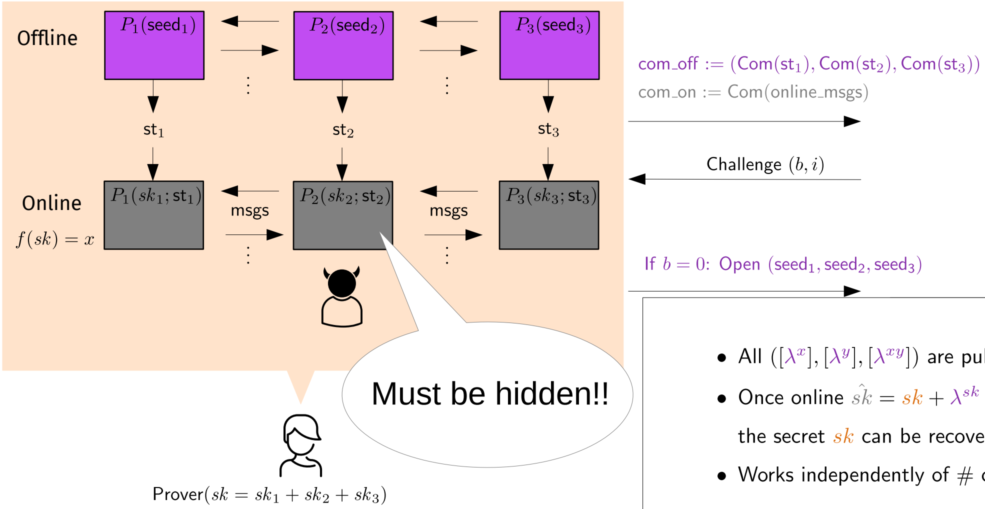
Attack II: Probing the unopened online phase



Attack II: Probing the unopened online phase



Attack II: Probing the unopened online phase



- All $([\lambda^x], [\lambda^y], [\lambda^{xy}])$ are public
- Once online $\hat{sk} = sk + \lambda^{sk}$ is probed, the secret sk can be recovered!!
- Works independently of # of parties!

Masking Picnic3

Our approach: Masking inside each party

Compute
$f(sk) = x$

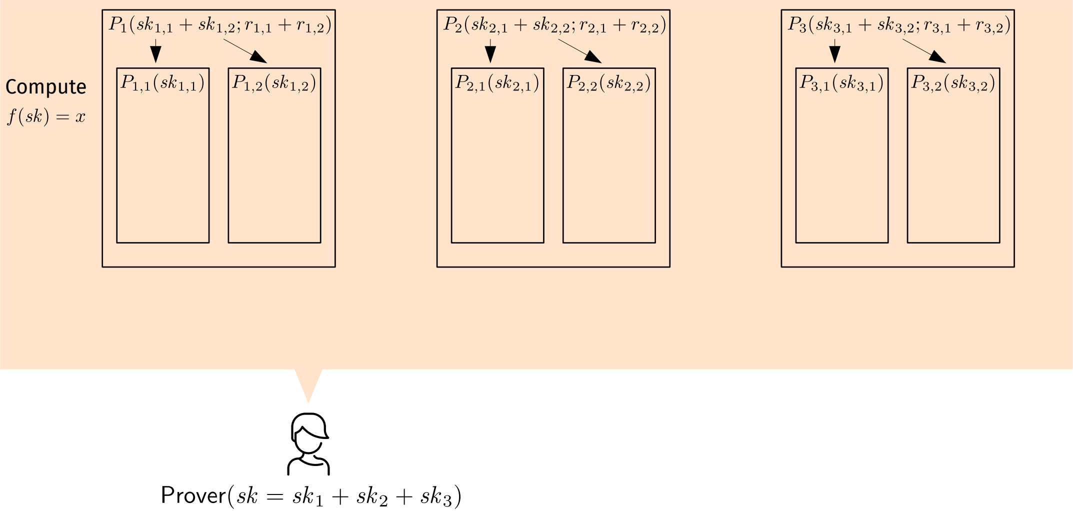
$P_1(sk_{1,1} + sk_{1,2}; r_{1,1} + r_{1,2})$

$P_2(sk_{2,1} + sk_{2,2}; r_{2,1} + r_{2,2})$

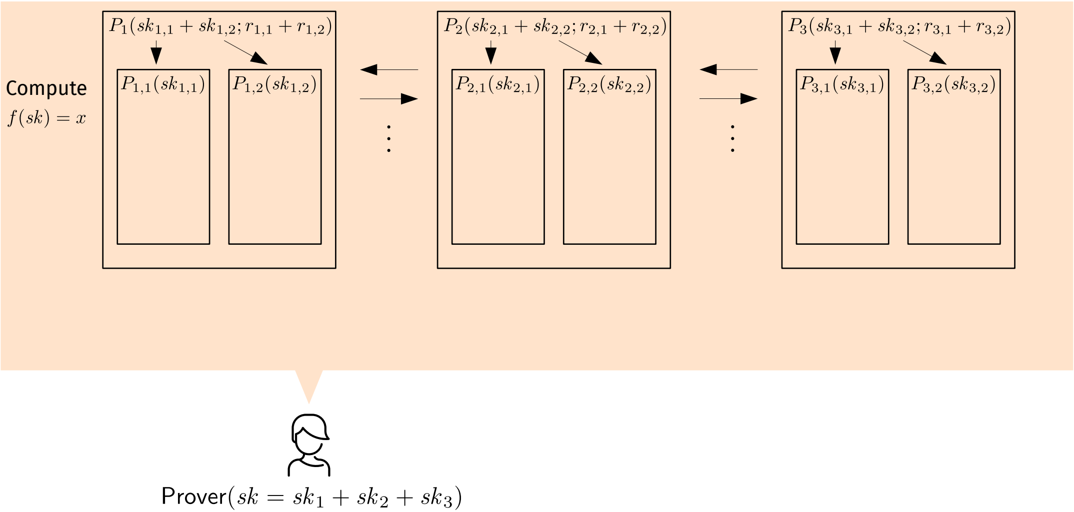
$P_3(sk_{3,1} + sk_{3,2}; r_{3,1} + r_{3,2})$

Prover$(sk = sk_1 + sk_2 + sk_3)$

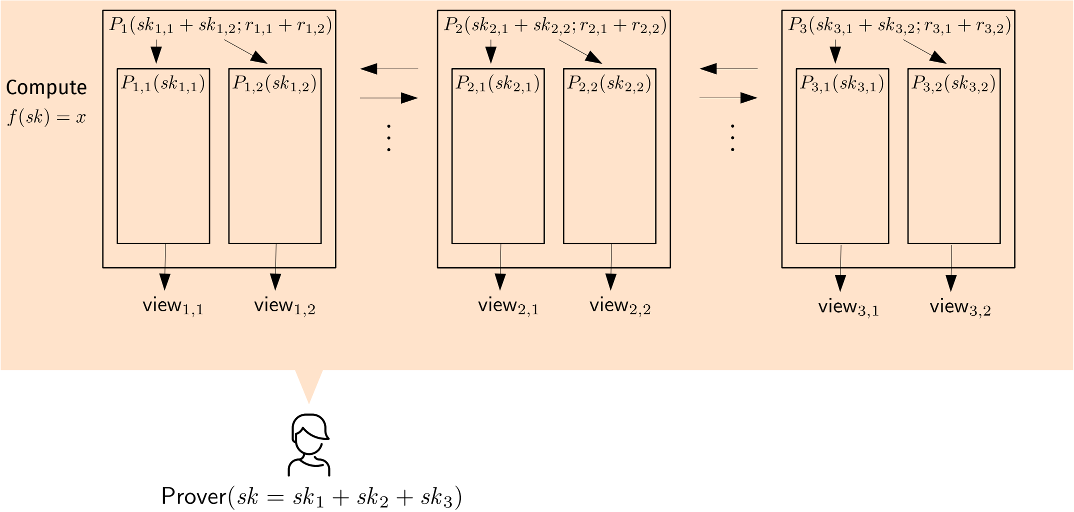
Our approach: Masking inside each party



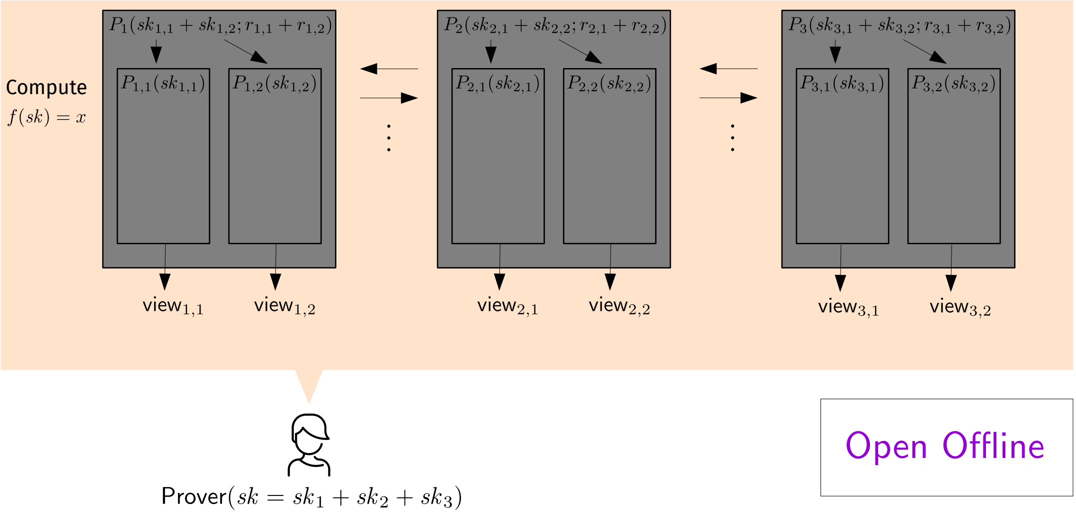
Our approach: Masking inside each party



Our approach: Masking inside each party

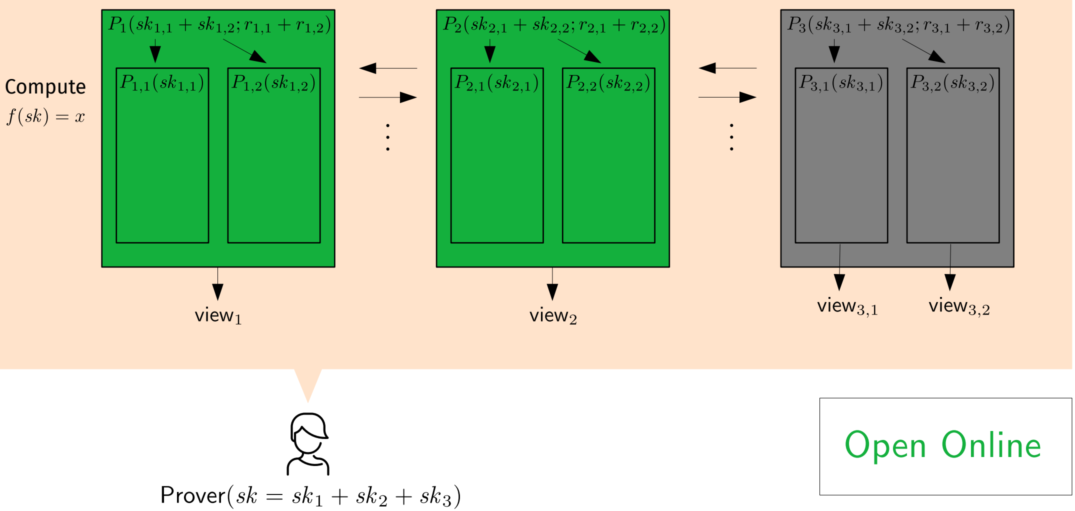


Our approach: Masking inside each party



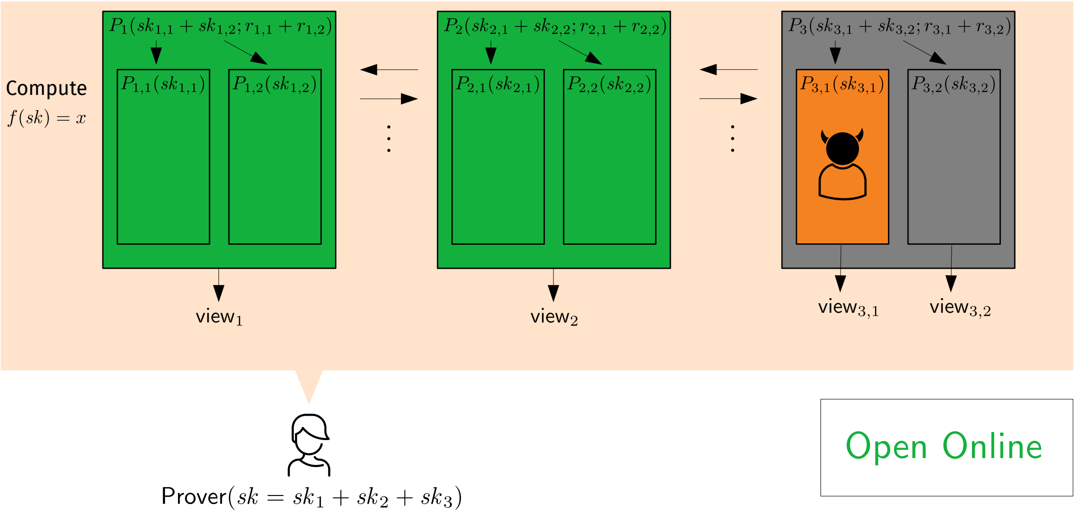
Open Offline

Our approach: Masking inside each party



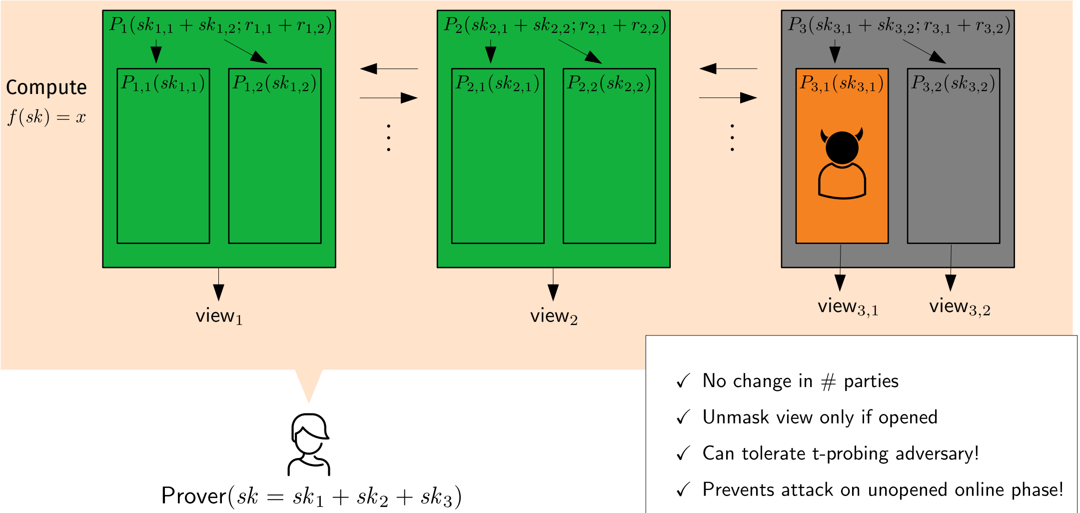
Open Online

Our approach: Masking inside each party



Open Online

Our approach: Masking inside each party



- ✓ No change in # parties
- ✓ Unmask view only if opened
- ✓ Can tolerate t-probing adversary!
- ✓ Prevents attack on unopened online phase!

Masking SHA-3

Masking seed expansion

- $[\text{tapes}_i] \leftarrow \text{SHA-3}([\text{seed}_i])$

Masking commitments

- $[\text{com\_off}_i] \leftarrow \text{SHA-3}([\text{st}_i])$
- $[\text{com\_on}] \leftarrow \text{SHA-3}([\text{online\_msgs}])$

Masking everything is expensive..

Heuristic options

- Some hash inputs that are unique per signature are not sensitive by regarding SHA-3 as a random oracle and if attacker only probes t bits of input.
- Commitment outputs are not sensitive
- Unmask / selectively mask half of the SHA-3 computations (without formal t-probing security)
- Empirically confirmed leakage resilience

Masking SHA-3

Masking seed expansion

- $[\text{tapes}_i] \leftarrow \text{SHA-3}([\text{seed}_i])$

Masking commitments

- $[\text{com\_off}_i] \leftarrow \text{SHA-3}([\text{st}_i])$
- $[\text{com\_on}] \leftarrow \text{SHA-3}([\text{online\_msgs}])$

Masking everything is expensive..

Heuristic options

- Some hash inputs that are unique per signature are not sensitive by regarding SHA-3 as a random oracle and if attacker only probes t bits of input.
- Commitment outputs are not sensitive
- Unmask / selectively mask half of the SHA-3 computations (without formal t-probing security)
- Empirically confirmed leakage resilience

Implementation & leakage analysis

Benchmarking for the First-order Protected Implementations

Picnic Masking	SHAKE Masking	Signing cycles	Hashing	Masking Overhead	Stack	Code	Random bytes(KB)
No	None	304	71%	1.00	32,460	121,349	0
Yes	None	460	50%	1.51	32,500	131,326	2,025
Yes	All-`SNI`	1663	86%	5.47	32,724	166,216	158,172
Yes	All-`DOM`	1289	81%	4.24	32,724	158,776	80,378
Yes	All-`IND`	856	72%	2.82	32,724	148.712	2,585
Yes	Selective	613	62%	2.01	32,460	148,712	2,025
Yes	Sel. Half	546	57%	1.80	32,460	148,712	2,025

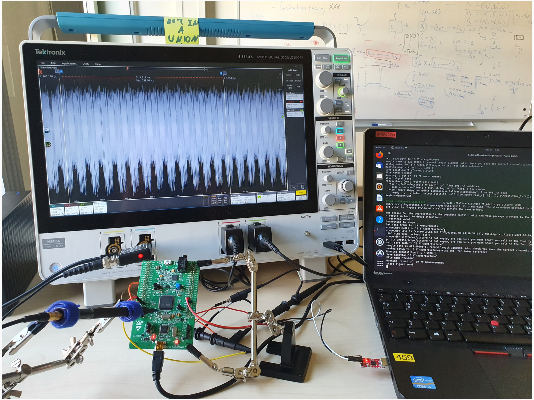
Table 1: Benchmarks in millions of Cortex-M4 cycles when $t = 1$.

Benchmarking for the First-order Protected Implementations

Picnic Masking	SHAKE Masking	Signing cycles	Hashing	Masking Overhead	Stack	Code	Random bytes(KB)
No	None	304	71%	1.00	32,460	121,349	0
Yes	None	460	50%	1.51	32,500	131,326	2,025
Yes	All-**SNI**	1663	86%	5.47	32,724	166,216	158,172
Yes	All-**DOM**	1289	81%	4.24	32,724	158,776	80,378
Yes	All-**IND**	856	72%	2.82	32,724	148.712	2,585
Yes	Selective	613	62%	2.01	32,460	148,712	2,025
Yes	Sel. Half	546	57%	1.80	32,460	148,712	2,025

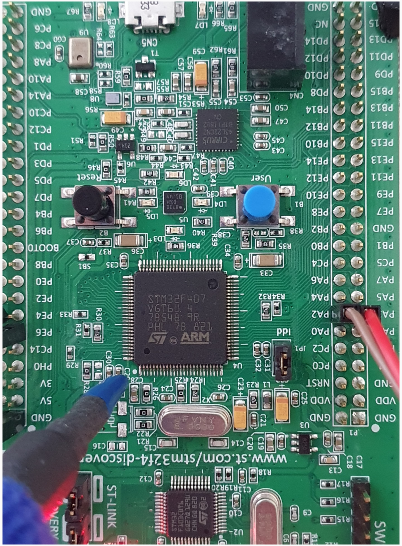
Table 1: Benchmarks in millions of Cortex-M4 cycles when $t = 1$.

A Practical Measurement Setup



- Capture: Tektronix MSO 6
- Short traces at 3.125 GS/s and long traces at 625 MS/s sampling rate
- Target device: STM32F4 Discovery board, Arm Cortex M4, operated at 168 MHz
- Source: EM emanations on a blocking cap (C29)

A Practical Measurement Setup



- Capture: Tektronix MSO 6
- Short traces at 3.125 GS/s and long traces at 625 MS/s sampling rate
- Target device: STM32F4 Discovery board, Arm Cortex M4, operated at 168 MHz
- Source: EM emanations on a blocking cap (C29)

Test Vector Leakage Assessment (TVLA)

A pass-fail test to decide if an implementation has exploitable leakage

- fixed-vs-random (FvR): to detect all possible first-order leakage.
- random-vs-random (RvR): to identify a specific exploitable leakage.

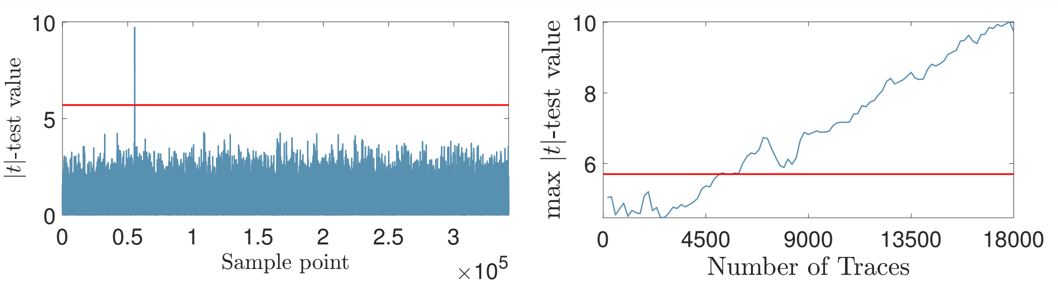
Goals

- Unprotected **Picnic3** is vulnerable (RvR)
- Protected **Picnic3** eliminates such vulnerabilities (FvR).

Attack I: Probing the opened online phase

$$\hat{x} = x + \lambda_1 + \cdots + \lambda_{N-1} + \lambda_N$$

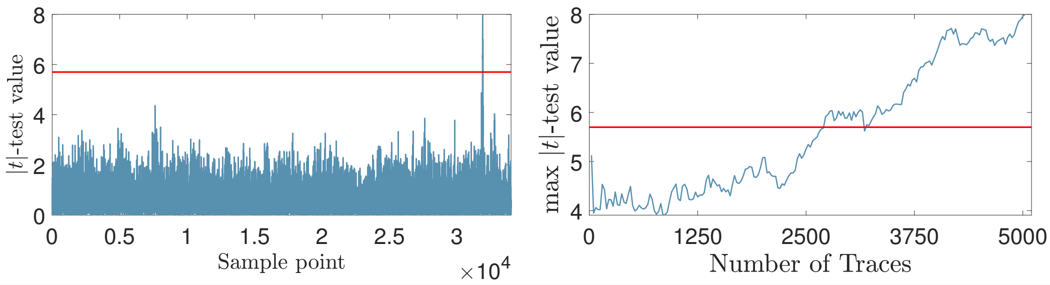
- Measurements from precomputation phase
- The leakage becomes clear after 6,000 traces.



Attack II: Probing the unopened online phase

$$\hat{x} = x + \lambda_1 + \cdots + \lambda_N$$

- Measurements from online simulation,
- The leakage becomes clear after 2,725 traces.



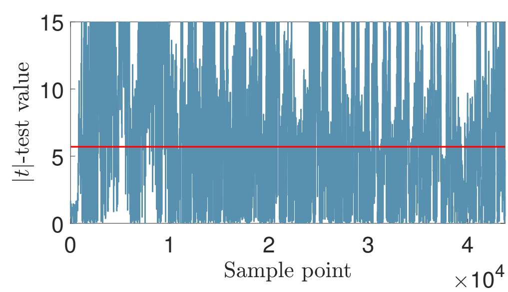
Leakage Analysis (FvR)

Masked SHA-3 (All-IND)

- 71 % of the calculation is hashing
- Fixing the mask value to a constant results in a leaking implementation with 2,000 traces.
- Randomizing the mask results in a non-leaky implementation with 10^6 traces.

Masked SHA-3 (All-IND)

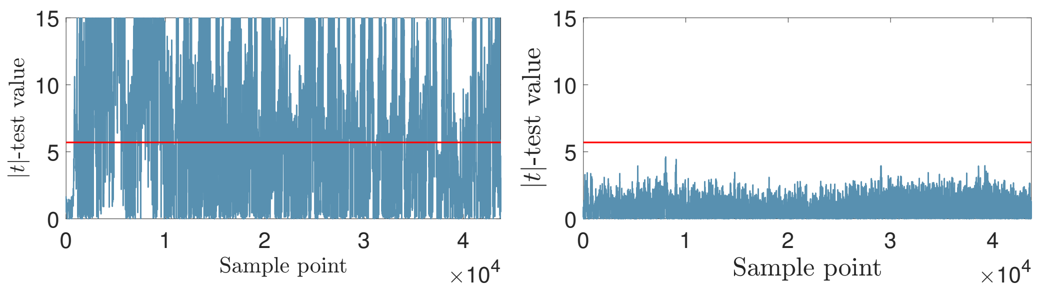
- 71 % of the calculation is hashing
- Fixing the mask value to a constant results in a leaking implementation with 2,000 traces.
- Randomizing the mask results in a non-leaky implementation with 10^6 traces.



Leakage Analysis (FvR)

Masked SHA-3 (All-IND)

- 71 % of the calculation is hashing
- Fixing the mask value to a constant results in a leaking implementation with 2,000 traces.
- Randomizing the mask results in a non-leaky implementation with 10^6 traces.



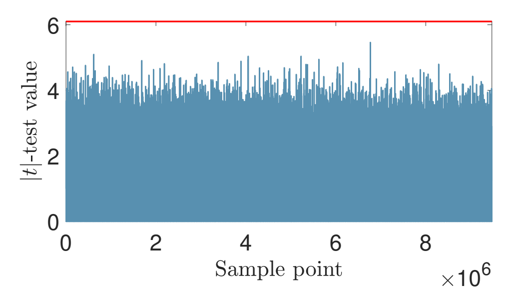
Leakage Analysis (FvR)

Masked Picnic3 (All-IND 4-round Masked SHA-3)

- Beginning of signature generation until the end of the first MPC instance
- Fixed vs Random *secret* key – fixed *message* – randomized *signature*.
- The $|t|$-value remains below threshold using 100,000 traces.
- Max $|t|$-value has a stable pattern.

Masked Picnic3 (All-IND 4-round Masked SHA-3)

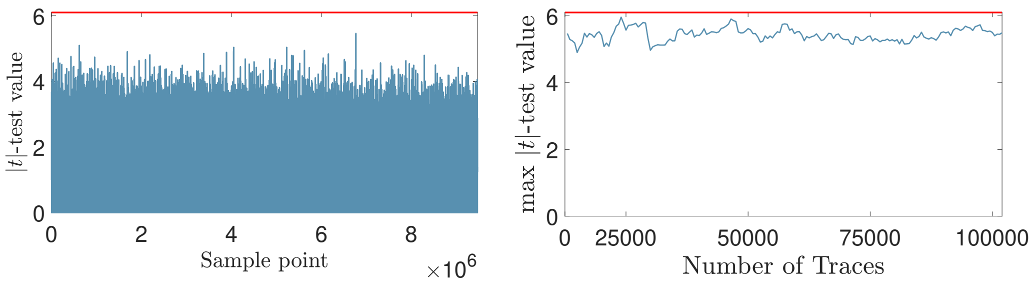
- Beginning of signature generation until the end of the first MPC instance
- Fixed vs Random *secret* key – fixed *message* – randomized *signature*.
- The $|t|$-value remains below threshold using 100,000 traces.
- Max $|t|$-value has a stable pattern.



Leakage Analysis (FvR)

Masked Picnic3 (All-IND 4-round Masked SHA-3)

- Beginning of signature generation until the end of the first MPC instance
- Fixed vs Random *secret* key – fixed *message* – randomized *signature*.
- The $|t|$-value remains below threshold using 100,000 traces.
- Max $|t|$-value has a stable pattern.



Takeaways

- Side-channel attacks against MPCitH with preprocessing is a real threat: as our two attacks demonstrate
- Generic masking countermeasures without breaking interoperability / increasing signature size
- Application to **Picnic3**: an overhead in the range of 1.80-5.47.
- Masked implementation of SHA-3: optimized with M4 assembly and supports a range of options, from slower but SNI-secure, to our much faster options.

Thank you!

Paper: `https://ia.cr/2021/735`
Implementation: `https://github.com/dkales/picnic_m4/`

Takeaways

- Side-channel attacks against MPCitH with preprocessing is a real threat: as our two attacks demonstrate
- Generic masking countermeasures without breaking interoperability / increasing signature size
- Application to **Picnic3**: an overhead in the range of 1.80-5.47.
- Masked implementation of SHA-3: optimized with M4 assembly and supports a range of options, from slower but SNI-secure, to our much faster options.

Thank you!

Paper: https://ia.cr/2021/735
Implementation: https://github.com/dkales/picnic_m4/

Takeaways

- Side-channel attacks against MPCitH with preprocessing is a real threat: as our two attacks demonstrate
- Generic masking countermeasures without breaking interoperability / increasing signature size
- Application to **Picnic3**: an overhead in the range of 1.80-5.47.
- Masked implementation of SHA-3: optimized with M4 assembly and supports a range of options, from slower but SNI-secure, to our much faster options.

Thank you!

Paper: `https://ia.cr/2021/735`

Implementation: `https://github.com/dkales/picnic_m4/`

Takeaways

- Side-channel attacks against MPCitH with preprocessing is a real threat: as our two attacks demonstrate
- Generic masking countermeasures without breaking interoperability / increasing signature size
- Application to **Picnic3**: an overhead in the range of 1.80-5.47.
- Masked implementation of SHA-3: optimized with M4 assembly and supports a range of options, from slower but SNI-secure, to our much faster options.

Thank you!

Paper: https://ia.cr/2021/735

Implementation: https://github.com/dkales/picnic_m4/

Takeaways

- Side-channel attacks against MPCitH with preprocessing is a real threat: as our two attacks demonstrate
- Generic masking countermeasures without breaking interoperability / increasing signature size
- Application to **Picnic3**: an overhead in the range of 1.80-5.47.
- Masked implementation of SHA-3: optimized with M4 assembly and supports a range of options, from slower but SNI-secure, to our much faster options.

Thank you!

Paper: https://ia.cr/2021/735
Implementation: https://github.com/dkales/picnic_m4/

References i

Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert.
maskVerif: Automated verification of higher-order masking in presence of physical defaults.
In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019, Part I*, volume 11735 of *LNCS*, pages 300–318. Springer, Heidelberg, September 2019.

Freepik.
Icons made by Freepik from Flaticon.com.
http://www.flaticon.com.

References ii

Irene Giacomelli, Jesper Madsen, and Claudio Orlandi.
ZKBoo: Faster zero-knowledge for Boolean circuits.
In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 1069–1083. USENIX Association, August 2016.

Tim Gellersen, Okan Seker, and Thomas Eisenbarth.
Differential power analysis of the Picnic signature scheme.
To appear at PQCrypto 2021. Cryptology ePrint Archive, Report 2020/267, 2020.
https://eprint.iacr.org/2020/267.

Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai.
Zero-knowledge from secure multiparty computation.
In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

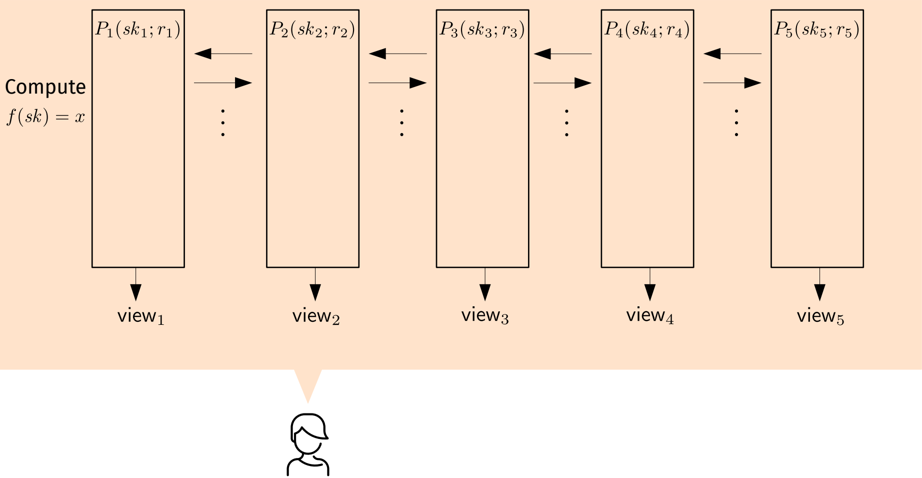
Yuval Ishai, Amit Sahai, and David Wagner.
Private circuits: Securing hardware against probing attacks.
In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, August 2003.

Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang.
Improved non-interactive zero knowledge with applications to post-quantum signatures.
In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 525–537. ACM Press, October 2018.

Daniel Kales and Greg Zaverucha.
Improving the performance of the Picnic signature scheme.
IACR TCHES, 2020(4):154–188, 2020.
https://tches.iacr.org/index.php/TCHES/article/view/8680.

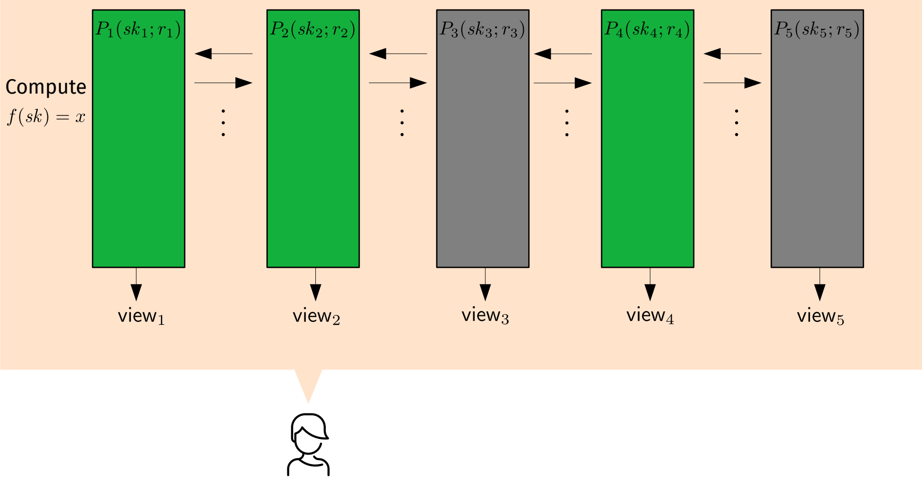
Okan Seker, Sebastian Berndt, Luca Wilke, and Thomas Eisenbarth.
SNI-in-the-head: Protecting MPC-in-the-head protocols against side-channel analysis.
In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 20*, pages 1033–1049. ACM Press, November 2020.

Previous countermeasure: SNI-in-the-head [SBWE20]



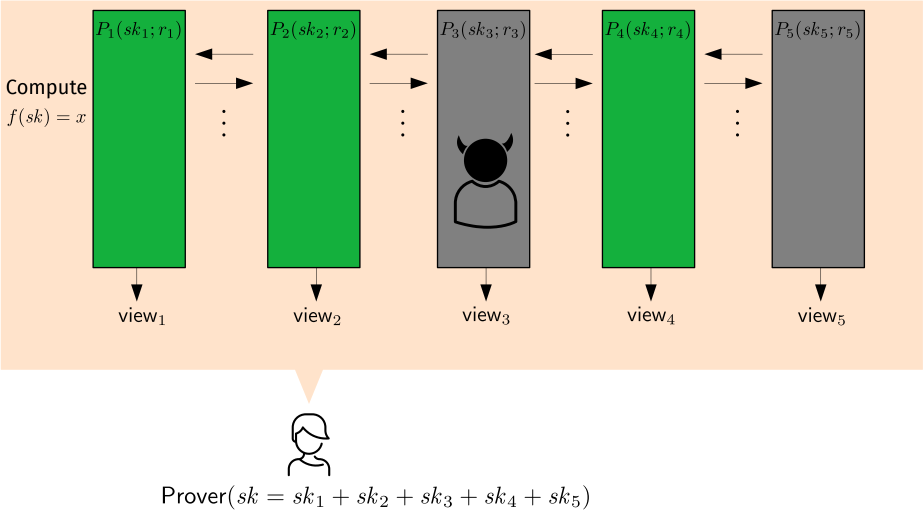
Prover$(sk = sk_1 + sk_2 + sk_3 + sk_4 + sk_5)$

Previous countermeasure: SNI-in-the-head [SBWE20]

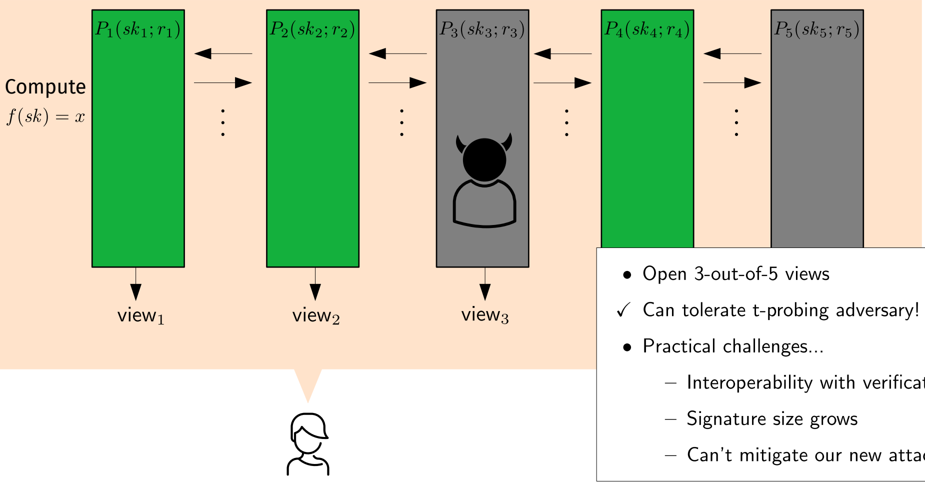


Prover$(sk = sk_1 + sk_2 + sk_3 + sk_4 + sk_5)$

Previous countermeasure: SNI-in-the-head [SBWE20]



Previous countermeasure: SNI-in-the-head [SBWE20]



Prover$(sk = sk_1 + sk_2 + sk_3 + sk_4 + sk_5)$

- Open 3-out-of-5 views
- ✓ Can tolerate t-probing adversary!
- Practical challenges...
  - Interoperability with verification
  - Signature size grows
  - Can't mitigate our new attack on MPCitH-PP

Masking MPC

SNI-secure Masked Online Multiplication

- Mask $[\hat{x}] := [x + \lambda^x]$ and $[\hat{y}] := [y + \lambda^y]$
- Each P_i computes

$$[z_i] = \delta_{1,i}\mathsf{SMul}([\hat{x}], [\hat{y}]) - \mathsf{SMul}([\hat{x}], [\lambda_i^y]) - \mathsf{SMul}([\hat{y}], [\lambda_i^x]) - [\lambda_i^{xy}]$$

✓ **SMul**: Standard SNI secure masked multiplier [ISW03]

✓ Never unmask $[\hat{x}]$ and $[\hat{y}]$ until the online phase can be safely revealed

✓ Applies to **any** MPCitH-PP-style signatures

✓ Securely **composable** with other gadgets thanks to the SNI property

Heuristics overview

- NI/SNI secure gadgets
- Input-sensitive, half-masked
- Output-sensitive, half-masked
- Unmasked