The MPC-in-the-head paradigm

Peter Scholl

Carsten Baum

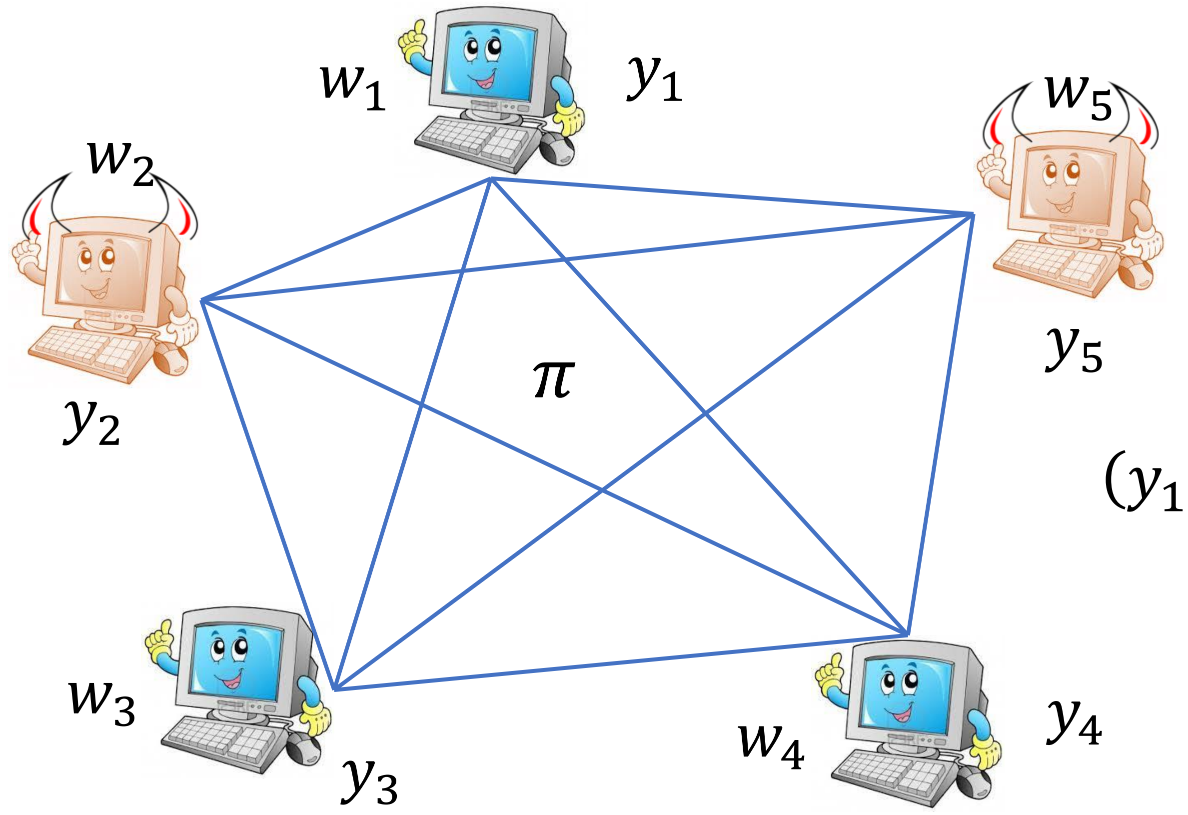
Schedule

1. Basics of MPC-in-the-head (now)
2. Signatures, Ligero & VOLEs
3. VOLE-in-the-head and FAEST

What we will cover in session 2

1. Signatures from MPC-in-the-head
2. The Ligero proof system
3. VOLEs

Recap: MPC



$(y_1, \dots, y_5) = C(w_1, \dots, w_5)$

Correctness: if parties learn the output, then it is y_i

$\boldsymbol{t_p}$**-Privacy:** no t_p parties can learn anything beyond their inputs and outputs from π

$\boldsymbol{t_r}$**-Robustness:** If $\leq t_r$ parties are actively corrupt, then all honest parties output y_i

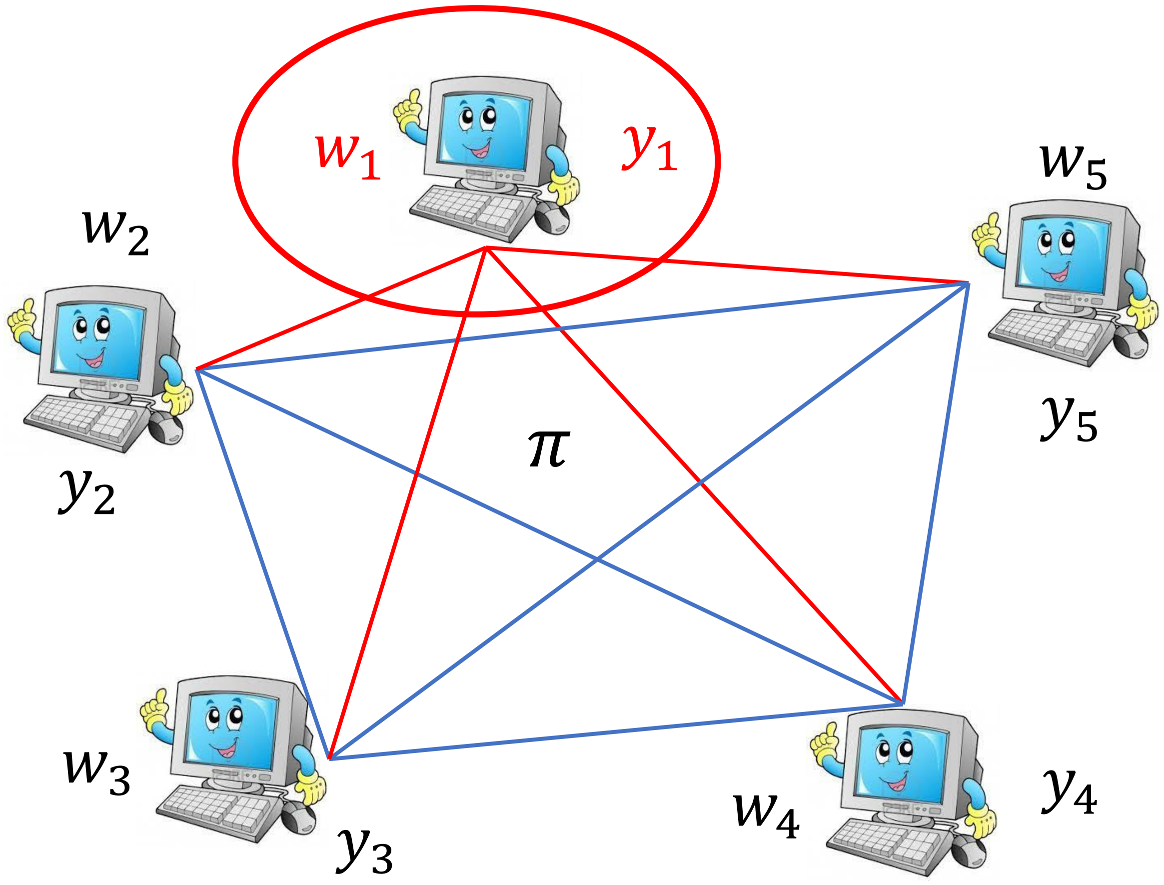
Views

View of P_1

1. All inputs of P_1
2. All outputs of P_1
3. All messages P_1 sent
4. All messages P_1 received

View of adversary

Views of all *corrupt* parties



Size of an MPC view in [KKW18]

Every party except P_N:

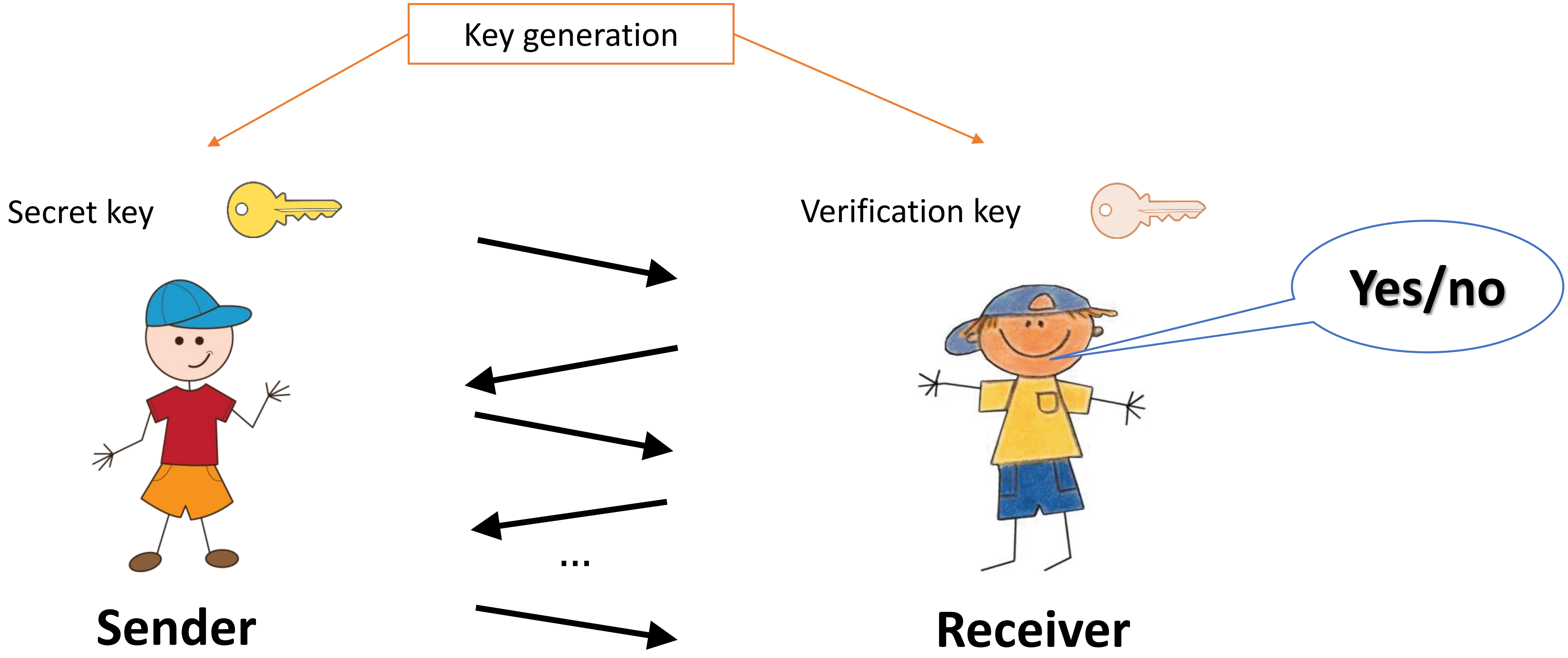
1. $seed_i$
2. For every multiplication: 2 shares from unopened party

P_N:

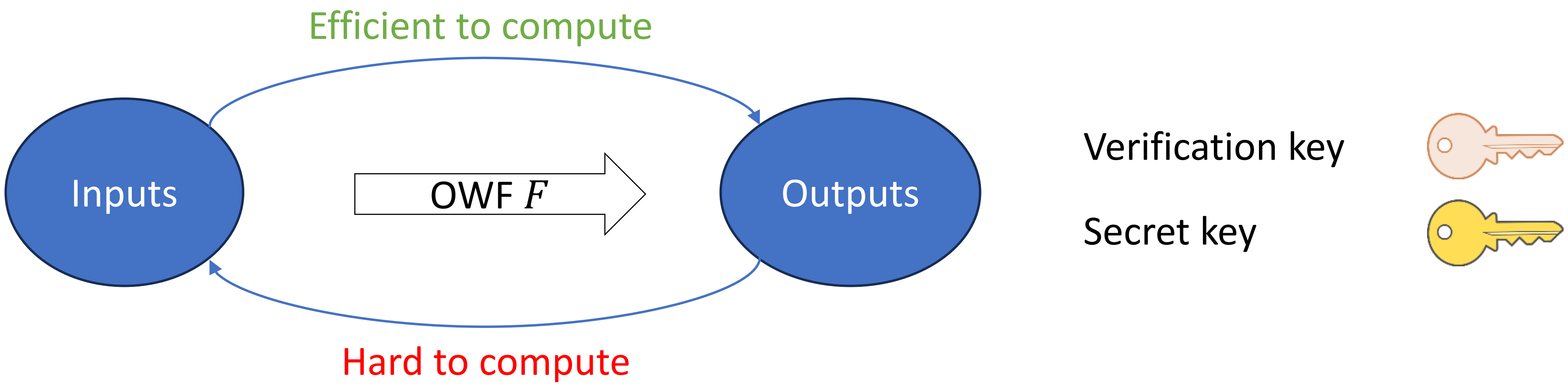
1. $seed_N$
2. 1 share per input, 1 share per triple
3. 2 shares for every multiplication

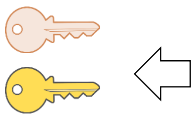
Proof size scales with $(\#inputs + \#multiplications) \cdot \log(|\mathbb{F}|)$

Identification



ZK proof + One-Way Function ⇒ Identification

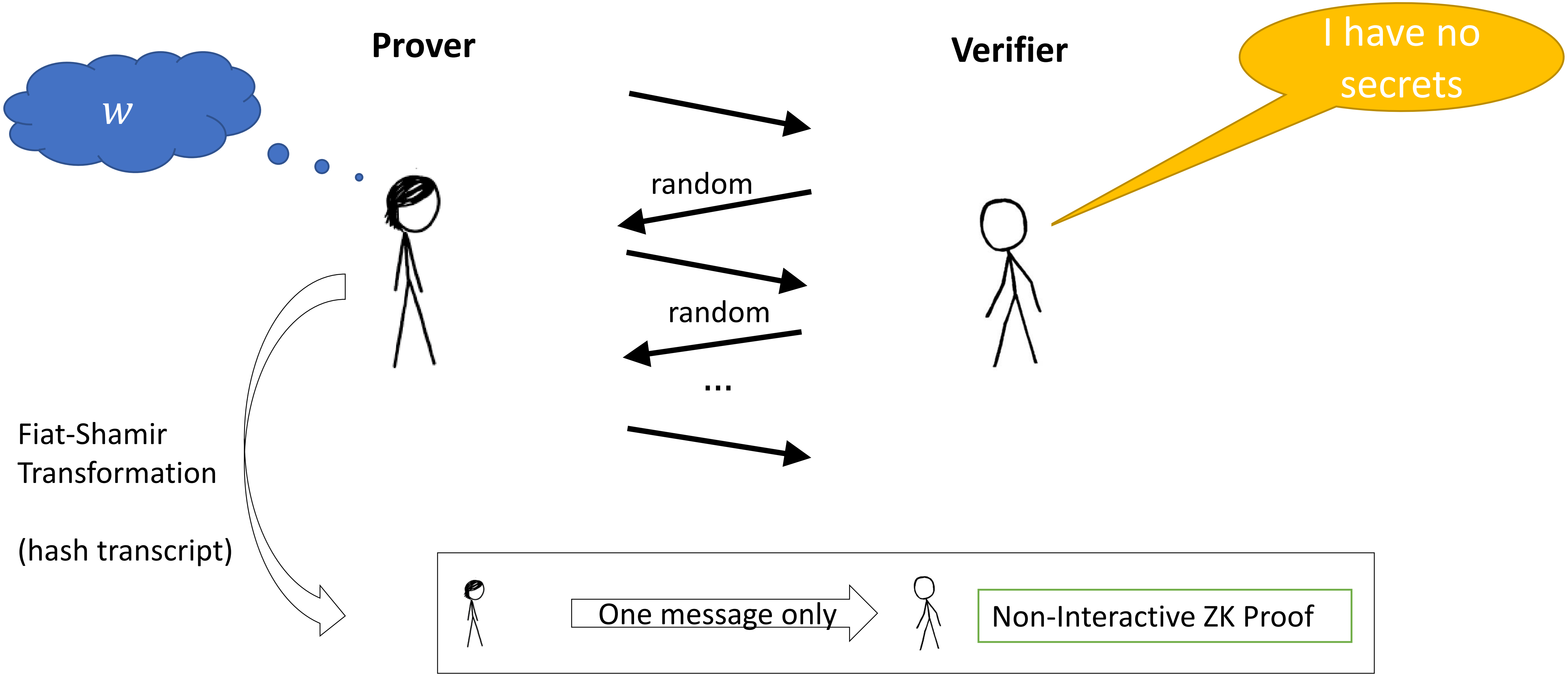




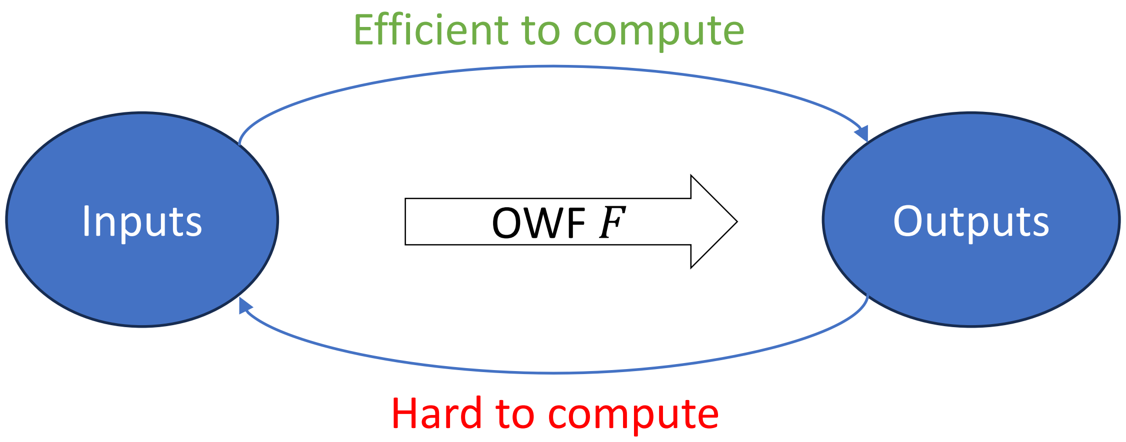
Key generation
Pick sk and compute $vk = F(sk)$

Identify
Generate interactive ZK proof π of knowledge of value sk s.th. $vk = F(sk)$

From Identification scheme to Signature



NIZK proof + One-Way Function ⇒ Signature



Verification key

Signing key

Key generation
Pick sk and compute $vk = F(sk)$

m σ

Sign
Generate non-interactive ZK proof π of value sk s.th. $vk = F(sk)$. Hash includes m & vk

σ, m, π

Verify
Verify ZK proof π for vk, m

NIZK proof + One-Way Function ⇒ Signature

Signing time = Prover time

Signature size = Proof size

Key generation
Pick sk and compute $= F(sk)$

m **Sign**
σ Generate non-interactive ZK proof π of value sk s.th. $vk = F(sk)$. Hash includes m & vk

σ, m, π **Verify**
Verify ZK proof π for vk, m

Attempt 1: Picnic

Proof size in [KKW18] etc. scales with
$(\#inputs + \#multiplications) \cdot \log(|\mathbb{F}|)$ } Signature size!

Use Block cipher as OWF with small input and #non-linear gates

E.g. LowMC cipher [ARS+15], used in the Picnic signature scheme

Cryptanalysis of Full LowMC and LowMC-M with Algebraic Techniques

Fukang Liu[1,2], Takanori Isobe[2,3,4], Willi Meier[5]

[1] East China Normal University, Shanghai, China
liufukangs@163.com
[2] University of Hyogo, Hyogo, Japan
[3] National Institute of Information and Communications Technology, Tokyo, Japan
[4] PRESTO, Japan Science and Technology Agency, Tokyo, Japan
takanori.isobe@ai.u-hyogo.ac.jp
[5] FHNW, Windisch, Switzerland
willimeier48@gmail.com

Abstract. In this paper, we revisit the difference enumeration technique for LowMC and develop new algebraic techniques to achieve efficient key-recovery attacks. In the original difference enumeration attack framework, an inevitable step is to precompute and store a set of intermediate state differences for efficient checking via the binary search. Our first observation is that Bar-On et al.'s general algebraic technique developed for SPNs with partial nonlinear layers can be utilized to fulfill the same task, which can make the memory complexity negligible as there is no need to store a huge set of state differences any more. Benefiting from this technique, we could significantly improve the attacks on LowMC when the block size is much larger than the key size and even break LowMC with such a kind of parameter. On the other hand, with our new key-recovery technique, we could significantly improve the time to retrieve the full key if given only a single pair of input and output messages together with the difference trail that they take, which was stated as an interesting question by Rechberger et al. at ToSC 2018. Combining both techniques, with only 2 chosen plaintexts, we could break 4 rounds of LowMC adopting a full S-Box layer with block size of 129, 192 and 255 bits, respectively, which are the 3 recommended parameters for Picnic3, an alternative third-round candidate in NIST's Post-Quantum Cryptography competition. We have to emphasize that our attacks do not indicate that Picnic3 is broken as the Picnic use-case is very different and an attacker cannot even freely choose 2 plaintexts to encrypt for a concrete LowMC instance. However, such parameters are deemed as secure in the latest LowMC. Moreover, much more rounds of seven instances of the backdoor cipher LowMC-M as proposed by Peyrin and Wang in CRYPTO 2020 can be broken without finding the backdoor by making full use of the allowed 2^{64} data. The above mentioned attacks are all achieved with negligible memory.

BBQ [DDOS19], Banquet [**B**DK+21], Limbo [DOT21]

Evaluate AES circuit over $\mathbb{F}_{2^8}$ (use [**B**N20] instead of [KKW18])

High-level description of the algorithm [edit]

1. `KeyExpansion` – round keys are derived from the cipher key using the AES key schedule. AES requires a separate 128-bit round key block for each round plus one more.
2. Initial round key addition:
    1. `AddRoundKey` – each byte of the state is combined with a byte of the round key using bitwise xor.
3. 9, 11 or 13 rounds:
    1. `SubBytes` – a non-linear substitution step where each byte is replaced with another according to a lookup table.
    2. `ShiftRows` – a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
    3. `MixColumns` – a linear mixing operation which operates on the columns of the state, combining the four bytes in each column.
    4. `AddRoundKey`
4. Final round (making 10, 12 or 14 rounds in total):
    1. `SubBytes`
    2. `ShiftRows`
    3. `AddRoundKey`

All operations except S-boxes are linear over $\mathbb{F}_2$

$SubBytes(x): x \rightarrow x^{-1}$ in $\mathbb{F}_{2^8}$ (and $0 \rightarrow 0$)

Protocol	N	M	τ	Sign (ms)	Ver (ms)	Size (bytes)
Picnic2	64	343	27	41.16	18.21	12 347
	16	252	36	10.42	5.00	13 831
Picnic3	16	252	36	5.33	4.03	12 466
SPHINCS$^+$-fast	-	-	-	14.42	1.74	16 976
SPHINCS$^+$-small	-	-	-	239.34	0.73	8 080
Banquet	16	-	41	6.36	4.86	19 776
	107	-	24	21.13	18.96	14 784
	255	-	21	43.81	40.11	13 284

Picnic, SPHINCS+ (using sha256simple) and Banquet for comparable parameter sizes and security levels (all run on Intel Xeon W-2133 CPU @ 3.60GHz) for NIST PQ L1 level

Other OWFs

Legendre PRF [BD20,Damgaard88]

Syndrome decoding [FJR22]

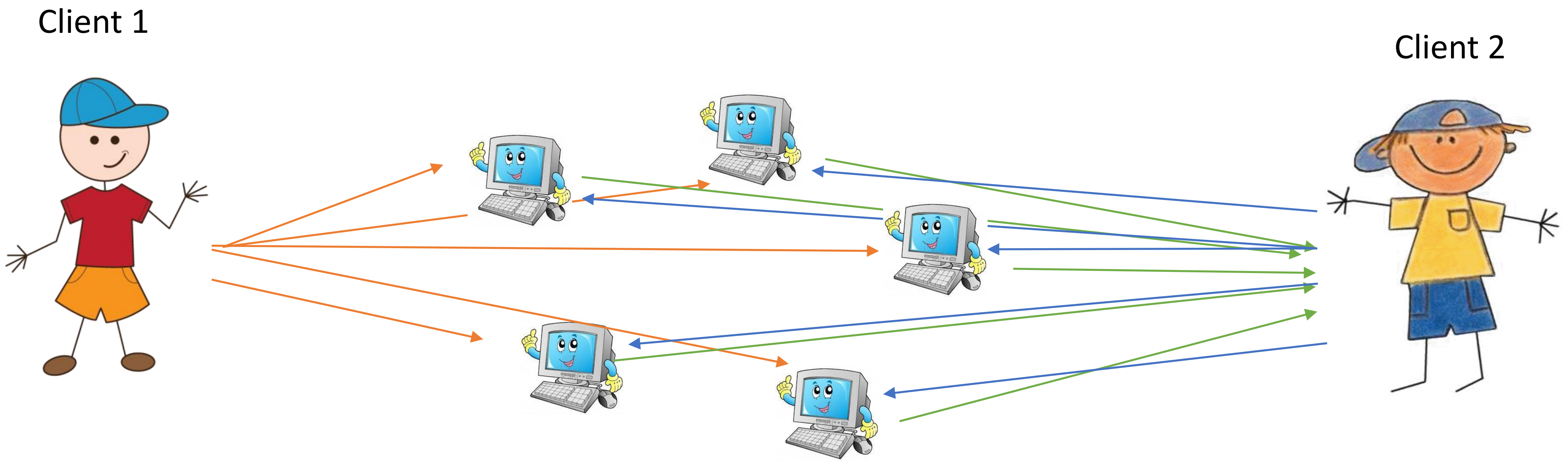
Multivariate Quadratic Polynomials [BFR23]

...

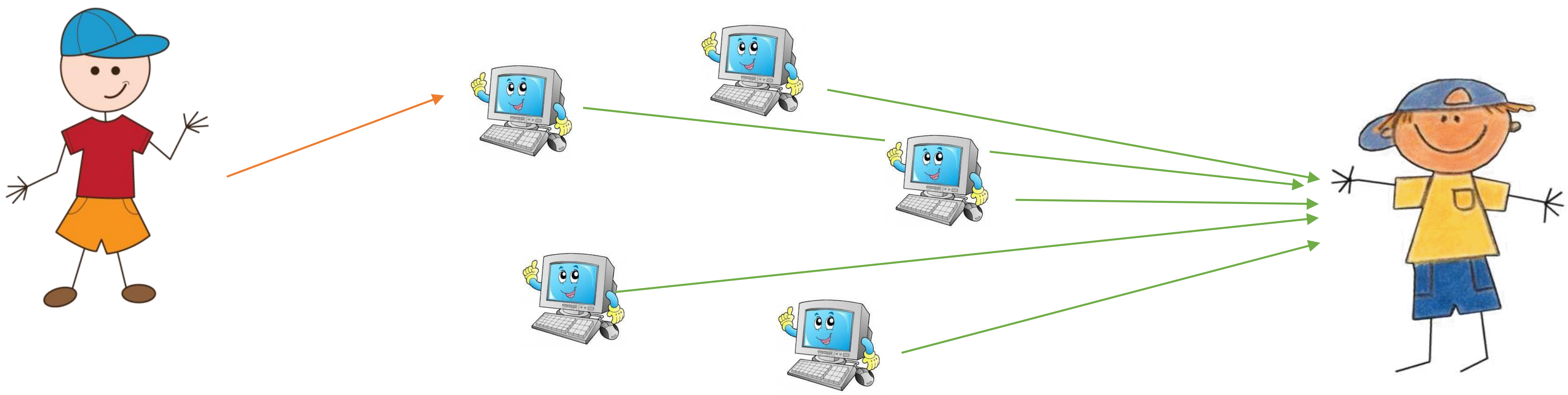
Ligero [AHIV17]

Getting below the circuit-size barrier

Modify the MPC scheme



Communication complexity of a proof



Proof size: $t \cdot \#input\ shares + \#parties$

Opened parties

Messages from MPC parties to verifier

Super-duper high level Ligero idea

Proof size: $\underbrace{t \cdot \#input\ shares}_{\text{Opened parties}} + \underbrace{\#parties}_{\text{Messages from MPC parties to verifier}}$

Let $|C| \approx |w|$. If $N = \sqrt{|C|}$, $t \approx \log(|C|)$ and $\#input\ shares \approx \sqrt{|C|}$, then communication $\tilde{O}(\sqrt{|C|})$

Use [DI06] protocol!

Shamir secret sharing

Secret $s \in \mathbb{F}$

Secrecy against t_p corruptions

Create $f \in \mathbb{F}[X], \deg(f) = t_p, f(0) = s$

$f(1)$ $\quad$ $f(2)$ $\quad$ $f(3)$ $\quad$ …

$\downarrow$ $\quad$ $\downarrow$ $\quad$ $\downarrow$

P_1 $\quad$ P_2 $\quad$ P_3

Observations

Create $f \in \mathbb{F}[X], \deg(f) = t_p, f(0) = s$

$f(1)$ $f(2)$ $f(3)$ …

$\downarrow$ $\downarrow$ $\downarrow$

P_1 P_2 P_3

1. Can reconstruct from any $t_p + 1$ shares
2. s uniformly random given t_p or less shares
3. Linearly homomorphic sharing (poly evaluation is homomorphism)

What if someone lies?

Create $f \in \mathbb{F}[X], \deg(f) = t_p, f(0) = s$

$f(1)$ → P_1

$f(2)$ → P_2

~~$f(3)$~~ → P_3

…

If $N > 3t_p$ shares, then can efficiently reconstruct with t_p faulty shares (Berlekamp-Welch)

Packed Shamir secret sharing

Secret $(s_1, \ldots, s_r) \in \mathbb{F}^r$

Secrecy against t_p corruptions

Create $f \in \mathbb{F}[X], \deg(f) = t_p + r, f(1-i) = s_i$

$f(1)$ → P_1

$f(2)$ → P_2

$f(3)$ → P_3

…

Notation

Packed sharing of $s_1, \ldots, s_r$ using poly of degree t: $(s_1, \ldots, s_r)_t$

Given $\alpha, \beta_1, \ldots, \beta_r, (a_1, \ldots, a_r)_{t_a}, (b_1, \ldots, b_r)_{t_b}$
parties can locally compute

$$(\alpha \cdot a_1 + b_1 + \beta_1, \ldots, \alpha a_r + b_r + \beta_r)_{\max(t_a, t_b)}$$

and

$$(a_1 \cdot b_1, \ldots, a_r \cdot b_r)_{t_a + t_b}$$

Share $O(N)$ secrets among N parties

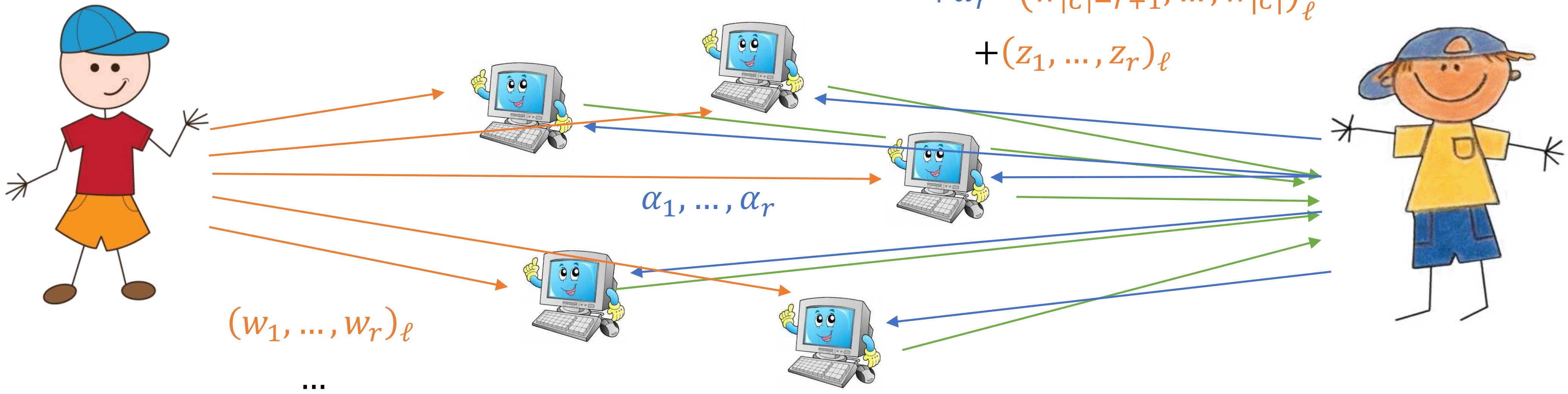
ZK if
$t = \ell - r$

$$(v_1, \dots, v_r)_\ell = \alpha_1 \cdot (w_1, \dots, w_r)_\ell + \dots + \alpha_r \cdot \left(w_{|C|-r+1}, \dots, w_{|C|}\right)_\ell + (z_1, \dots, z_r)_\ell$$

Let $r = N = \sqrt{|C|}$

Client 1

Client 2



$(w_1, \dots, w_r)_\ell$

…

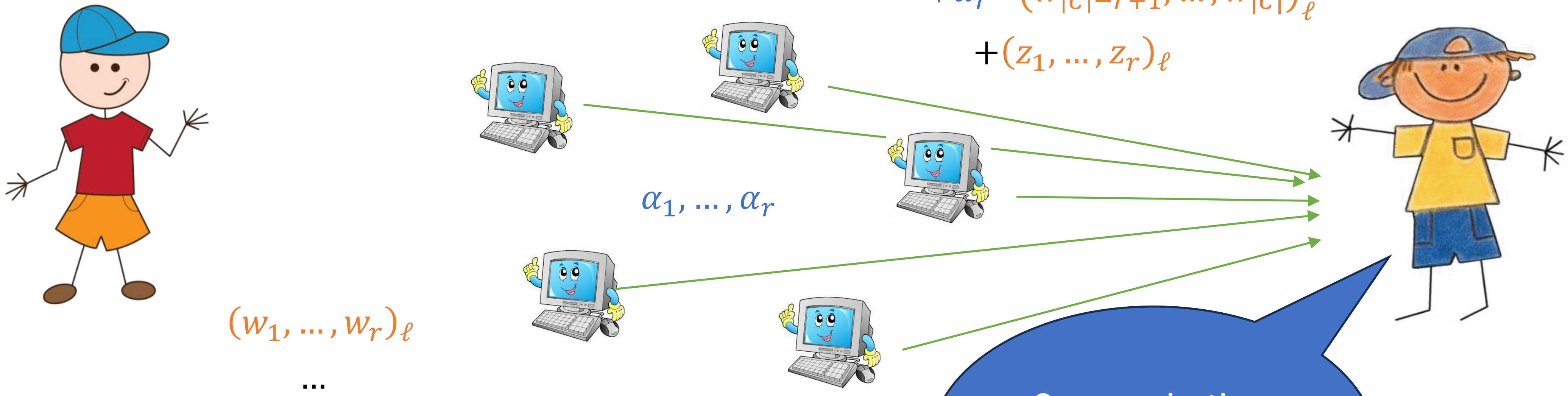
$\left(w_{|C|-r+1}, \dots, w_{|C|}\right)_\ell$

$(z_1, \dots, z_r)_\ell$

Share $O(N)$ secrets among N parties

Let $r = N = \sqrt{|C|}$

Client 1



Client 2

$(v_1, \dots, v_r)_\ell$

$=$

$\alpha_1 \cdot (w_1, \dots, w_r)_\ell$

$+ \cdots +$

$+\alpha_r \cdot \left(w_{|C|-r+1}, \dots, w_{|C|}\right)_\ell$

$+(z_1, \dots, z_r)_\ell$

$\alpha_1, \dots, \alpha_r$

$(w_1, \dots, w_r)_\ell$

…

$\left(w_{|C|-r+1}, \dots, w_{|C|}\right)_\ell$

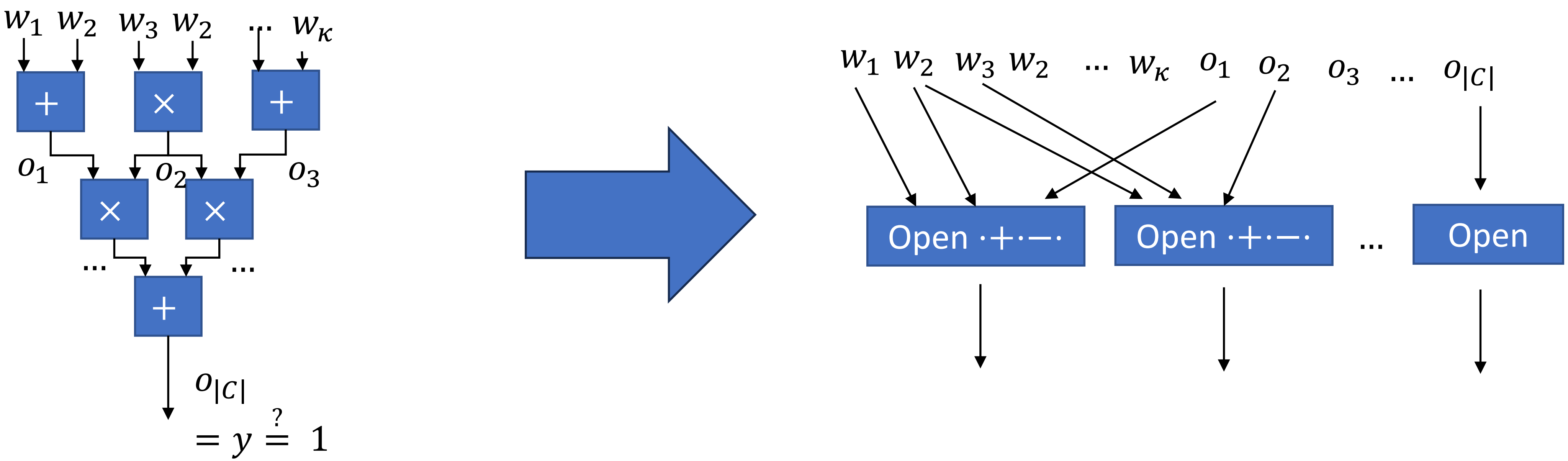
$(z_1, \dots, z_r)_\ell$

t large enough:
All $(\cdot, \dots, \cdot)_\ell$
decodable

Communication:
$t \cdot r + N$ elements
in $\mathbb{F}$

Extending the witness

$$(x, w) \in R_L \Leftrightarrow C(w) = 1$$

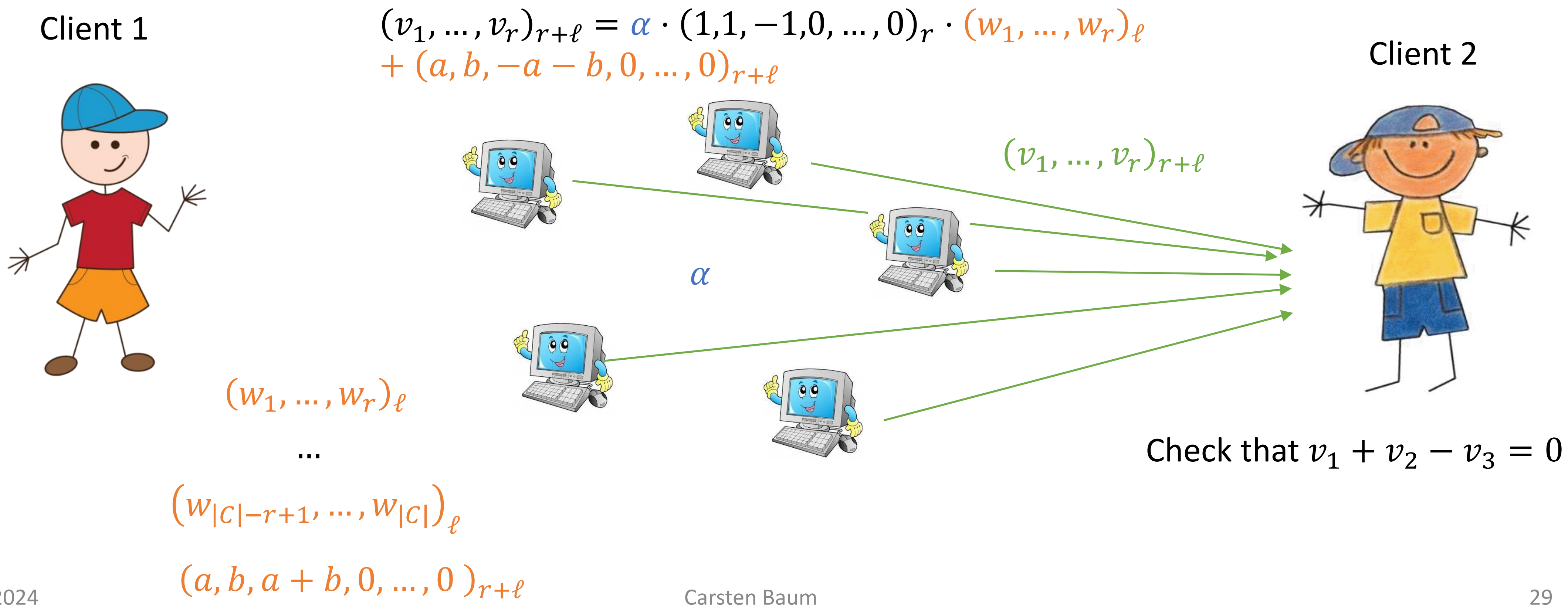


Circuit evaluation

Circuit consistency check

Check linear relations

For $(w_1, \ldots, w_r)_\ell$ check that $w_1 + w_2 = w_3 \leftrightarrow w_1 + w_2 - w_3 = 0$



Check linear relations

Cost:

1. Secret sharing of $(a, b, -a - b, 0, \ldots, 0\ )_{r+\ell}$ by prover
2. Sending $(v_1, \ldots, v_r)_{r+\ell}$ to verifier

One can show:

One sharing by prover and message to verifier enough to check *any number of linear relations*

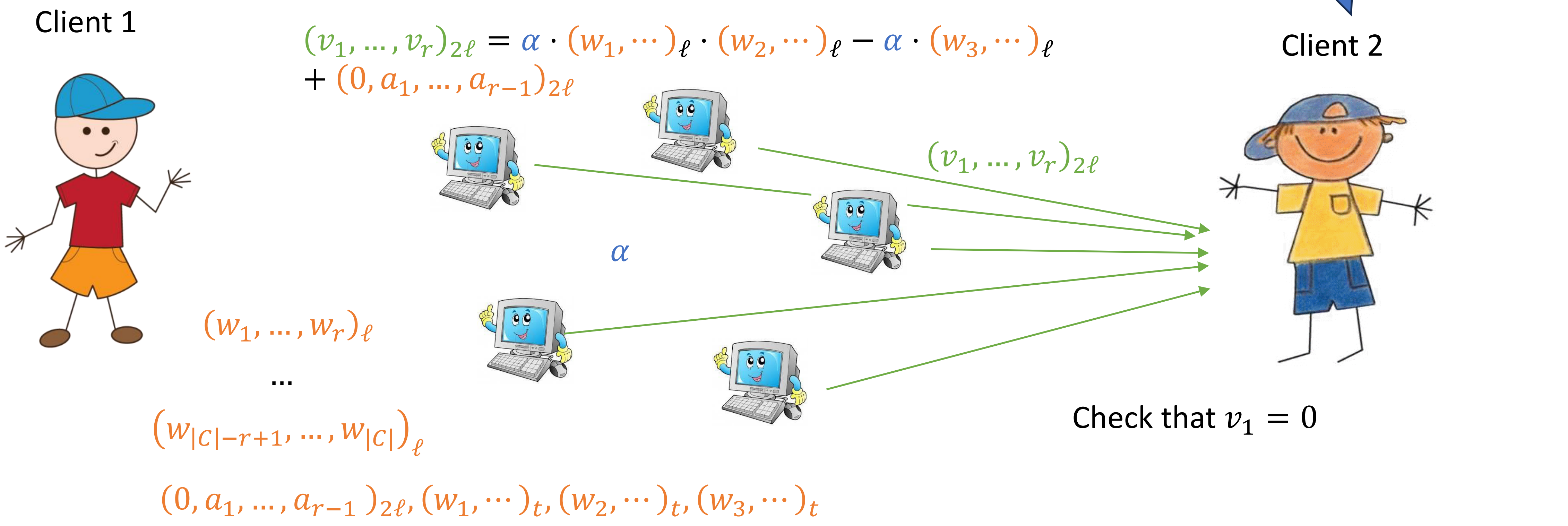
For the experts:

1. Use random linear combination
2. Use blinding vector that sums to 0

Check multiplicative relations

$(w_1, \cdots)_\ell, (w_2, \cdots)_\ell, (w_3, \cdots)_\ell$
consistent with $(w_1, \ldots, w_r)_\ell$?

For $(w_1, \ldots, w_r)_t$ check that $w_1 \cdot w_2 = w_3 \leftrightarrow w_1 \cdot w_2 - w_3 = 0$



Check multiplicative relations

Cost:

1. Sharing of $(w_1, \cdots)_\ell, (w_2, \cdots)_\ell, (w_3, \cdots)_\ell, (0, a_1, \dots, a_{r-1})_{2\ell}$
2. Sending $(v_1, \dots, v_r)_{2\ell}$ to verifier

One can show:

Can verify r multiplications with 4 sharings + 1 opening + linear check

(for $\mathrm{r}^2 = |C|$ multiplications we need $O(\sqrt{|C|})$ sharings + 1 opening + linear check)

Further reading

[CDG+17] Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D. & Zaverucha, G. (2017). Post-quantum zero-knowledge and signatures from symmetric-key primitives.

[DDOS19] de Saint Guilhem, C. D., De Meyer, L., Orsini, E., & Smart, N. P. (2019, August). BBQ: using AES in picnic signatures.

[**B**DK+21] Baum, C., de Saint Guilhem, C. D., Kales, D., Orsini, E., Scholl, P., & Zaverucha, G. (2021, May). Banquet: short and fast signatures from AES.

[DOT21] Delpech de Saint Guilhem, C., Orsini, E., & Tanguy, T. (2021, November). Limbo: efficient zero-knowledge MPCitH-based arguments.

[BD20] Beullens, W., & Delpech de Saint Guilhem, C. (2020, April). LegRoast: Efficient post-quantum signatures from the Legendre PRF.

[FJR22] Feneuil, T., Joux, A., & Rivain, M. (2022, August). Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs.

[BFR23] Benadjila, R., Feneuil, T., & Rivain, M. (2023). MQ on my mind: Post-quantum signatures from the non-structured multivariate quadratic problem.

[AHIV17] Ames, S., Hazay, C., Ishai, Y., & Venkitasubramaniam, M. (2017, October). Ligero: Lightweight sublinear arguments without a trusted setup.