Circuit Amortization Friendly Encodings and Their Application to Statistically Secure Multiparty Computation

Anders Dalskov, **Eysa Lee**, and Eduardo Soria-Vazquez

Aarhus University Northeastern University

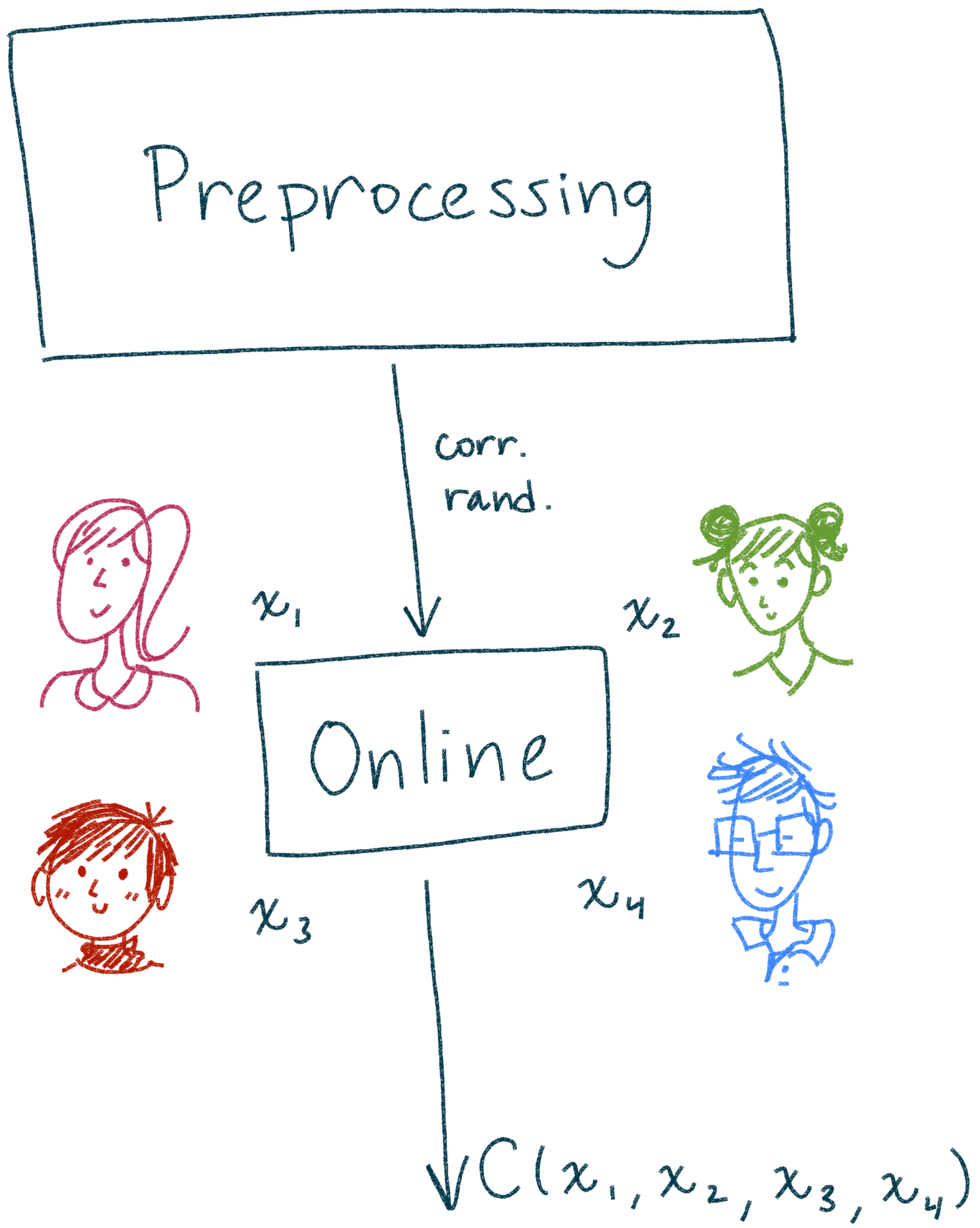
MPC setting in this talk

- Mixed computation

want to support switching between arithmetic and boolean circuits

- Preprocessing phase
- Active adversary corrupting up to $t < n/3$ parties
- Security with abort

can apply standard techniques to get guaranteed output delivery, but not a focus of this work



Shamir secret sharing over a field

Can interpolate with D+1 points, so typically set D = t

Sharing a secret s:

- Sample a degree D polynomial $p(x)$ where $p(0) = s$
- Evaluate $p(x)$ at public $x_1, \ldots, x_n$
- Distribute $y_i = p(x_i)$ to party i

Reconstructing a secret:

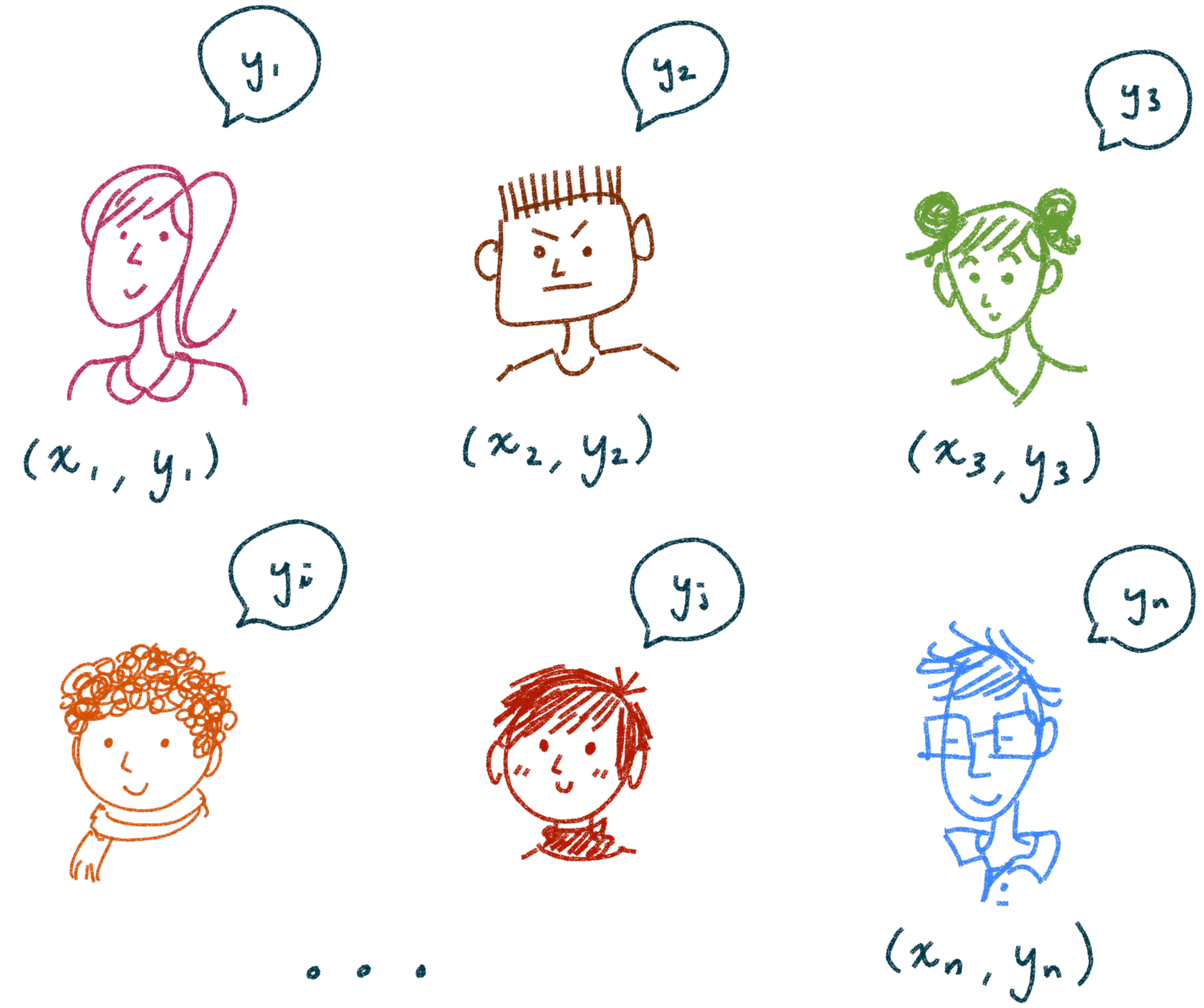
- Each party i announces their share (x_i, y_i)
- Parties compute $s = p(0)$ using

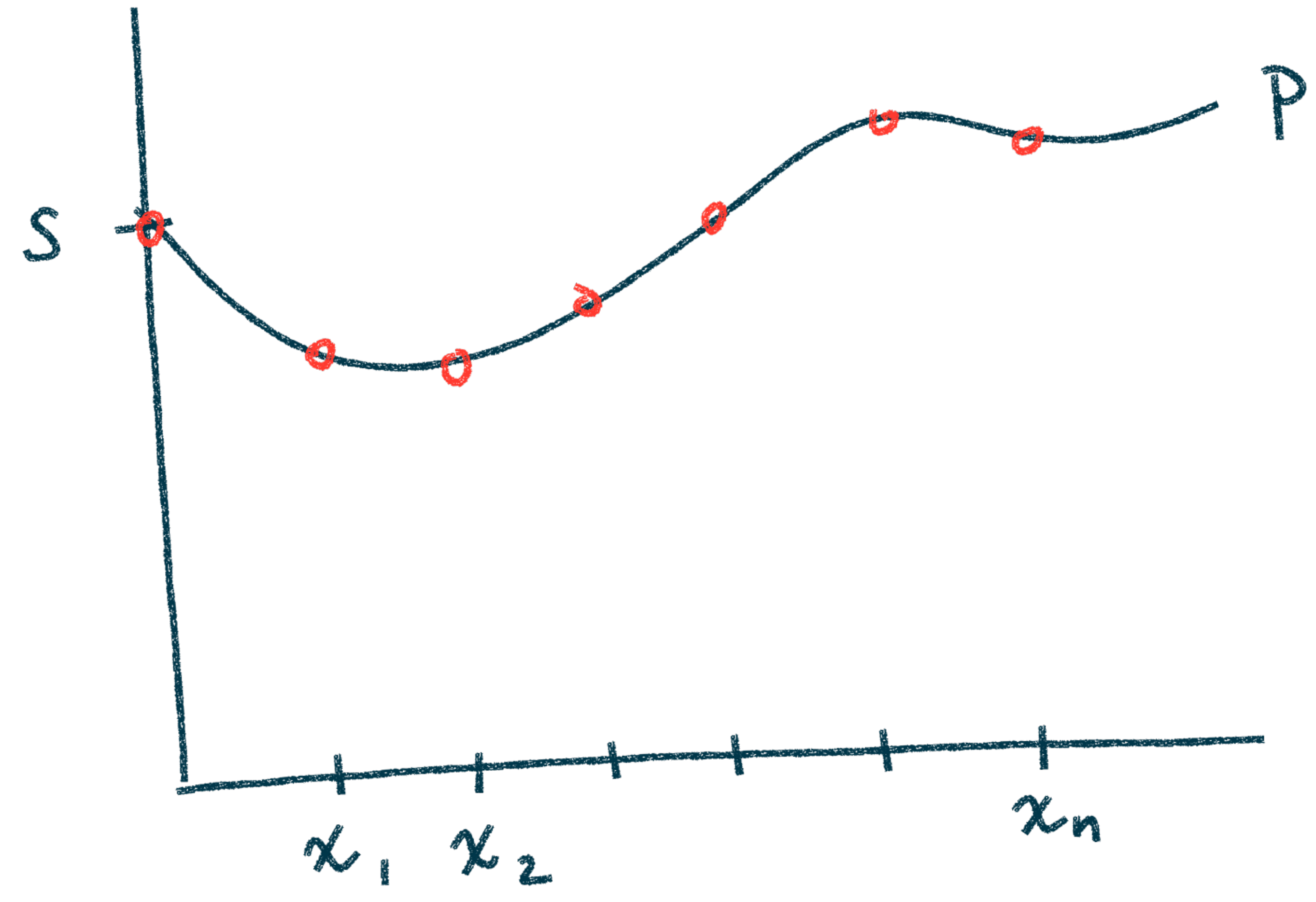
Check p(x) is of degree D

$$p(x) = \sum_{j=1}^{n} y_j \cdot l_j(x)$$

Inverse always exists, as $x_j \neq x_i$ and we are on a field

$$\text{where } l_j(x) = \prod_{i=1, i\neq j}^{n} (x - x_i) \cdot (x_j - x_i)^{-1}$$





MPC from Shamir secret sharing

Let $\langle a \rangle_D$ denote the sharing of a by a polynomial of degree D

Linearity: $\langle a \rangle_D + \langle b \rangle_D = \langle a + b \rangle_D$

Multiplication: $\langle a \rangle_D \cdot \langle b \rangle_D = \langle a \cdot b \rangle_{2D}$

We need $\geq 2D+1$ parties to reconstruct!

can't do this forever...

Using preprocessed double shares $\left(\langle r \rangle_t, \langle r \rangle_{2t} \right)$, we can reduce the degree as follows:

1. Locally compute $\langle a \cdot b \rangle_{2t} = \langle a \rangle_t \cdot \langle b \rangle_t$
2. Publicly reconstruct $\langle z \rangle_{2t} = \langle a \cdot b \rangle_{2t} - \langle r \rangle_{2t}$
3. Locally compute $\langle a \cdot b \rangle_t = z + \langle r \rangle_t$

Shamir secret sharing over $\mathbb{Z}_{2^k}$?

Sharing a secret s:

- Sample a degree D polynomial $p(x)$ where $p(0) = s$
- Evaluate $p(x)$ at public $x_1, x_2, \ldots, x_n$
- Distribute $y_i = p(x_i)$ to party i

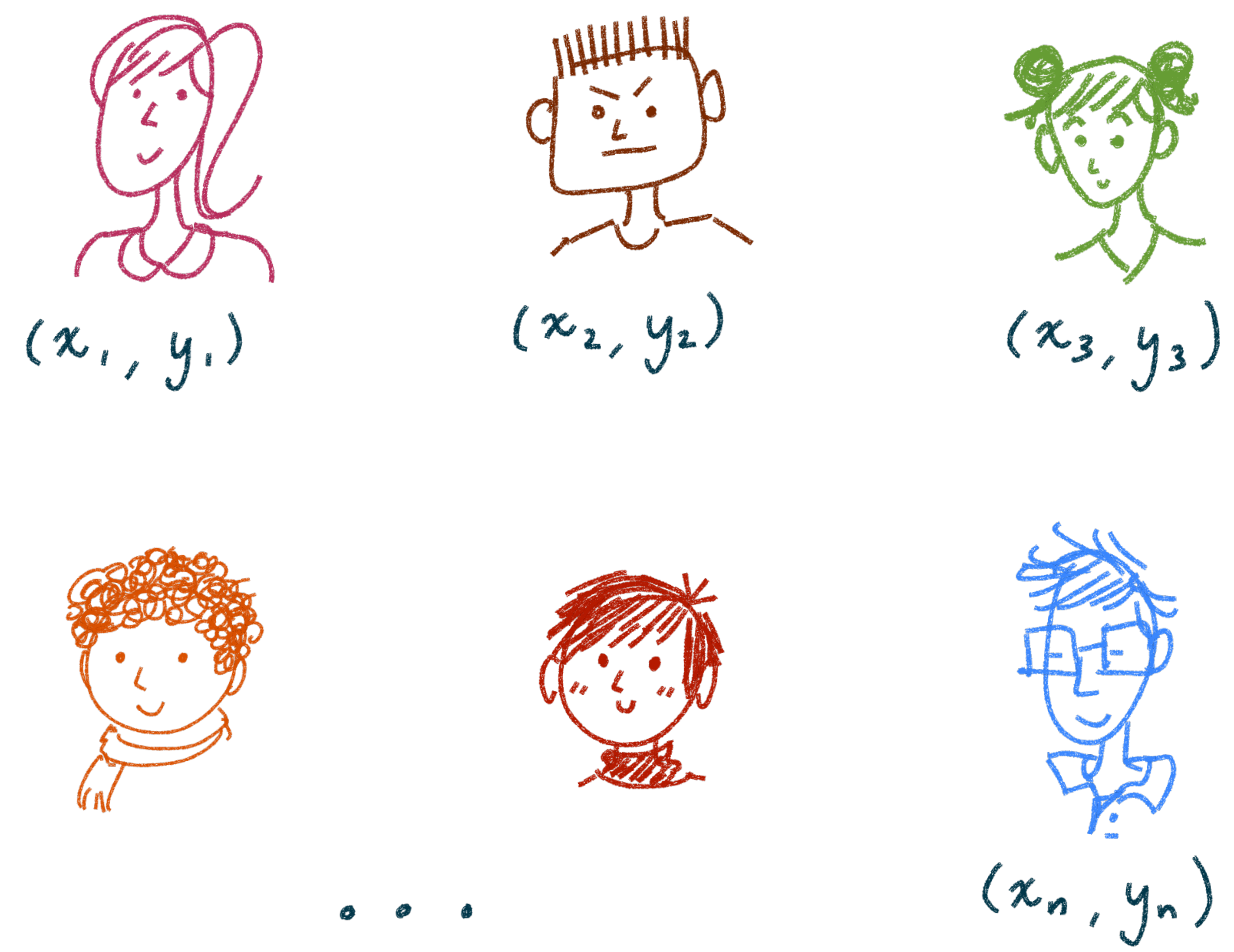
Doesn't work for arbitrary number of parties!

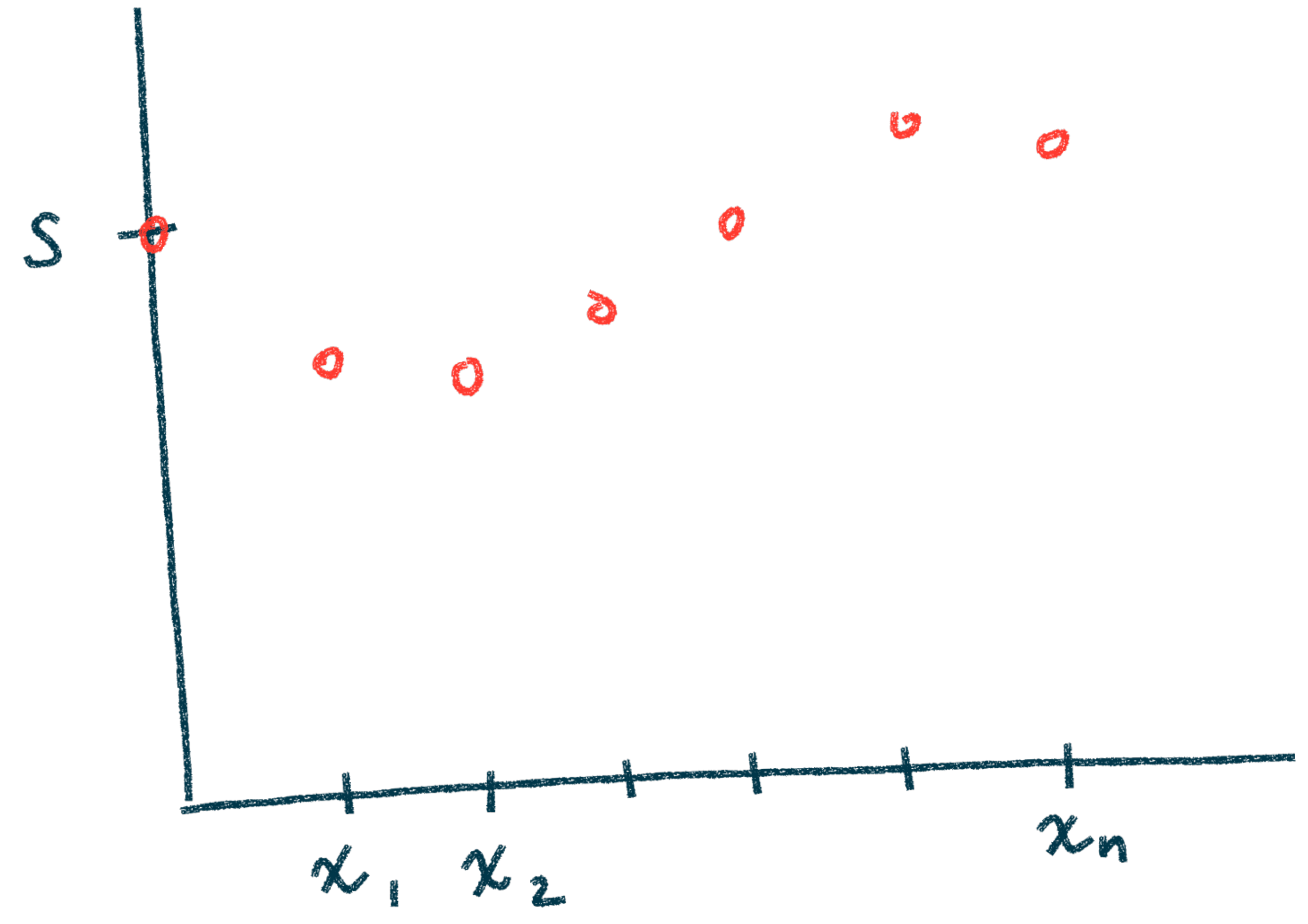
Reconstructing a secret:

- Each party i announces their share (x_i, y_i)
- Parties compute $s = p(0)$ using

$$p(x) = \sum_{j=1}^{n} y_j \cdot l_j(x)$$

$$\text{where } l_j(x) = \prod_{i=1, i \neq j}^{n} (x - x_i) \cdot (x_j - x_i)^{-1}$$





Shamir secret sharing over $\mathbb{Z}_{2^k}$?

Sharing a secret s:

- Sample a degree D polynomial $p(x)$ where $p(0) = s$
- Evaluate $p(x)$ at public $x_1, x_2, \ldots, x_n$
- Distribute $y_i = p(x_i)$ to party i

Doesn't work for arbitrary number of parties!

Reconstructing a secret:

- Each party i announces their share (x_i, y_i)
- Parties compute $s = p(0)$ using

$$p(x) = \sum_{j=1}^{n} y_j \cdot l_j(x)$$

$$\text{where } l_j(x) = \prod_{i=1, i \neq j}^{n} (x - x_i) \cdot \boxed{(x_j - x_i)^{-1}}$$

Why? $\mathbb{Z}_{2^k}$ has no subset $S = \{x_1, \ldots, x_n\} \subset \mathbb{Z}_{2^k}$ where all pairwise differences are invertible for $n > 2$.

Proof: If $n > 2$, then $\exists\, x_i, x_j$ s.t. $2 \mid (x_i - x_j)$

Hence $2^{k-1} \cdot (x_i - x_j) = 0$

$\Rightarrow (x_i - x_j)$ is not invertible

Such a set satisfying this is called an exceptional set

How to overcome this?

a degree-d Galois extension of $\mathbb{Z}_{2^k}$ is a Galois Ring $GR(2^k, d)$

Basics of Galois Rings

A Galois Ring $GR(p^k, d)$ is of the form

$$R = \mathbb{Z}_{p^k}[X]/(h(X))$$

where p is prime, $k \geq 1$, and $h(X) \in \mathbb{Z}_{p^k}[X]$ is a monic polynomial of degree $d \geq 1$ such that its reduction modulo p yields an irreducible polynomial in $\mathbb{F}_p[X]$

Arbitrary element $a \in GR(p^k, d)$ can be described as $a = a_{d-1} \cdot \xi^{d-1} + \ldots + a_1 \cdot \xi + a_0$ where $a_i \in \mathbb{Z}_{p^k}$ and ξ is a root of $h(X)$.

Additive representation

Some properties of Galois Rings:

- $GR(p, d) = \mathbb{F}_{p^d}$
- All zero divisors of $R = GR(p^k, d)$ constitute R's only maximal ideal, (p)
- $GR(p^k, d)$ has exceptional sets of size p^d

We can do polynomial interpolation!

MPC over $\mathbb{Z}_{2^k}$ via Galois Rings [ACDEY19]

[ACDEY19] adapts the protocol of [BH08] to $\mathbb{Z}_{2^k}$ using Galois Rings

additive representation of elem $a \in GR(p^k, d)$

$$a = a_{d-1}\xi^{d-1} + \ldots + a_1\xi + a_0$$

where $a_i \in \mathbb{Z}_{p^k}$ and ξ is a root of $h(X)$

- Exceptional set for $R = GR(2^k, d)$ is size 2^d, so set $d = \log_2(n+1)$
- Natural embedding $\iota : \mathbb{Z}_{2^k} \hookrightarrow R$

  can think of as const polynomial

  - Just look at any $x \in \mathbb{Z}_{2^k}$ as an element of R: Shamir secret sharing works over R
- [BH08] perfectly translates to R! But overhead of extension degree means:
  - Communication complexity multiplied by a factor of d

    poly w/ d coefficients from $\mathbb{Z}_{2^k}$
  - Computational complexity of multiplication is quadratic in d

[ACDEY 19] "Efficient Information-Theoretic Secure Multiparty Computation over $\mathbb{Z}_{p^k}$ via Galois Rings"
Mark Abspoel, Ronald Cramer, Ivan Damgård, Daniel Escudero, and Chen Yuan

[BH 08] "Perfectly Secure MPC with linear communication complexity"
Zuzana Beerliová-Trubíniová and Martin Hirt.

Our main contribution:

Better encodings from $\mathbb{Z}_{2^k}$ to $GR(2^k, d)$

[ACDEY19]: 1 mult in GR $\Rightarrow$ 1 mult in $\mathbb{Z}_{2^k}$

In [ACDEY19], elements $a, b \in \mathbb{Z}_{2^k}$ are encoded into $R = GR(2^k, d)$ according to the natural inclusion $\iota : \mathbb{Z}_{2^k} \hookrightarrow R$

Multiplication makes use of double shares $\left(\langle \iota(r) \rangle_t \,, \langle \iota(r) \rangle_{2t}\right)$:

1. Locally compute $\langle \iota(a \cdot b) \rangle_{2t} = \langle \iota(a) \rangle_t \cdot \langle \iota(b) \rangle_t$
2. Publicly reconstruct $\langle z \rangle_{2t} = \langle \iota(a \cdot b) \rangle_{2t} - \langle \iota(r) \rangle_{2t}$
3. Locally compute $\langle \iota(a \cdot b) \rangle_t = z + \langle \iota(r) \rangle_t$

Can we use the extension degree $d = \log(n)$ to compute more expressive circuits?

[ACDEY19]: 1 mult in GR $\Rightarrow$ 1 mult in $\mathbb{Z}_{2^k}$

In [ACDEY19], elements $a, b \in \mathbb{Z}_{2^k}$ are encoded into $R = GR(2^k, d)$ according to the natural inclusion $\iota : \mathbb{Z}_{2^k} \hookrightarrow R$

Multiplication makes use of double shares $\left(\langle \iota(r) \rangle_t\,, \langle \iota(r) \rangle_{2t}\right)$:

1. Locally compute $\langle \iota(a \cdot b) \rangle_{2t} = \langle \iota(a) \rangle_t \cdot \langle \iota(b) \rangle_t$
2. Publicly reconstruct $\langle z \rangle_{2t} = \langle \iota(a \cdot b) \rangle_{2t} - \langle \iota(r) \rangle_{2t}$
3. Locally compute $\langle \iota(a \cdot b) \rangle_t = z + \langle \iota(r) \rangle_t$

Can we use the extension degree $d = \log(n)$ to compute more expressive circuits?

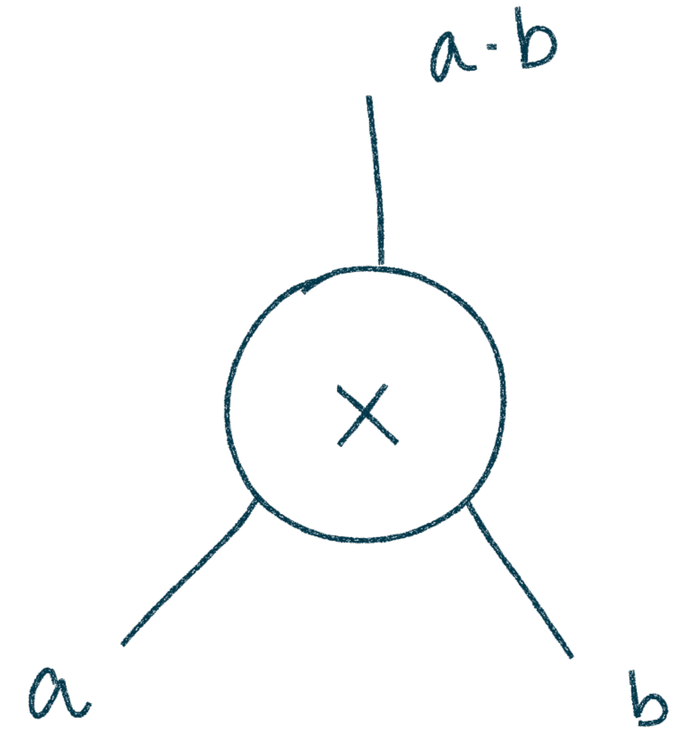
Yes! We substitute $\iota : \mathbb{Z}_{2^k} \hookrightarrow R$ for encodings $E : (\mathbb{Z}_{2^k})^\delta \hookrightarrow R$, where $\delta \leq d$

Translating multiplications in GR to circuits in $\mathbb{Z}_{2^k}$

Let $E_{in} : (\mathbb{Z}_{2^k})^{\delta_1} \to R$ and $E_{out} : (\mathbb{Z}_{2^k})^{\delta_2} \to R$ be two $\mathbb{Z}_{2^k}$-linear maps such that

$$E_{in}(\overrightarrow{x}) \cdot E_{in}(\overrightarrow{y}) + E_{out}(\vec{r}) = E_{out}(C(\overrightarrow{x}, \overrightarrow{y}) + \vec{r})$$

Where $C(a_1, \ldots, a_{2\delta_1}) = (b_1, \ldots, b_{\delta_2})$ is our desired subcircuit (arithmetic over $\mathbb{Z}_{2^k}$)

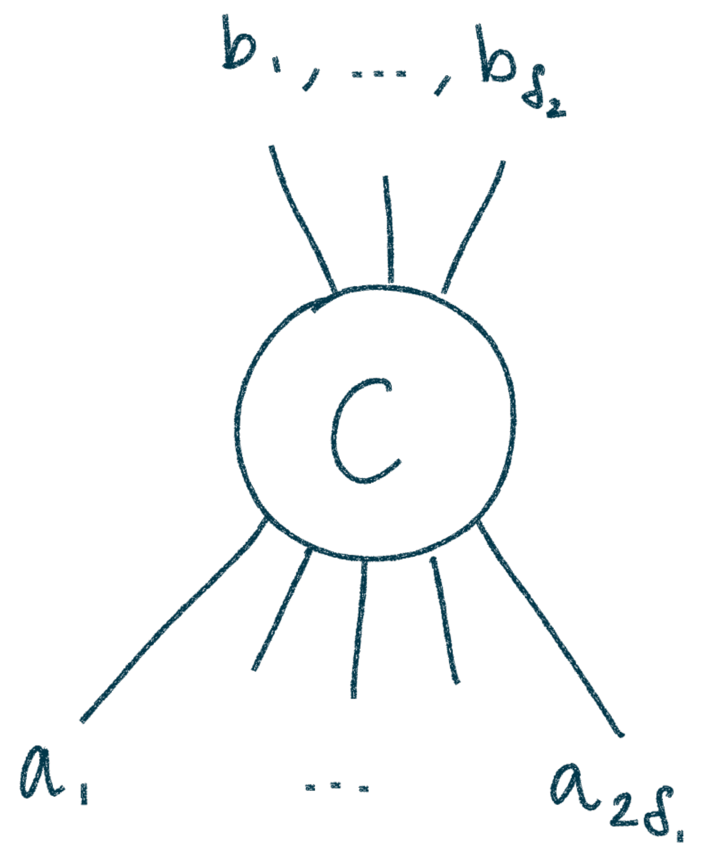


Given double shares $\left(\langle E_{in}(\vec{r})\rangle_t \,, \langle E_{out}(\vec{r})\rangle_{2t}\right)$, where $\vec{r} \in (\mathbb{Z}_{2^k})^{\delta_2}$:

1. Locally compute $\langle E_{in}(\overrightarrow{x}) \cdot E_{in}(\overrightarrow{y})\rangle_{2t} = \langle E_{in}(\overrightarrow{x})\rangle_t \cdot \langle E_{in}(\overrightarrow{y})\rangle_t$
2. Publicly reconstruct $\langle z\rangle_{2t} = \langle E_{in}(\overrightarrow{x}) \cdot E_{in}(\overrightarrow{y})\rangle_{2t} - \langle E_{out}(\vec{r})\rangle_{2t}$
3. Locally compute $\langle E_{in}(C(\overrightarrow{x}, \overrightarrow{y}))\rangle_t = E_{in}(E_{out}^{-1}(z)) + \langle E_{in}(\vec{r})\rangle_t$

Same outline as before, but with encodings

With $E_{in}(x)$, $E_{in}(y)$, and double shares, can compute $E_{in}(C(x,y))$



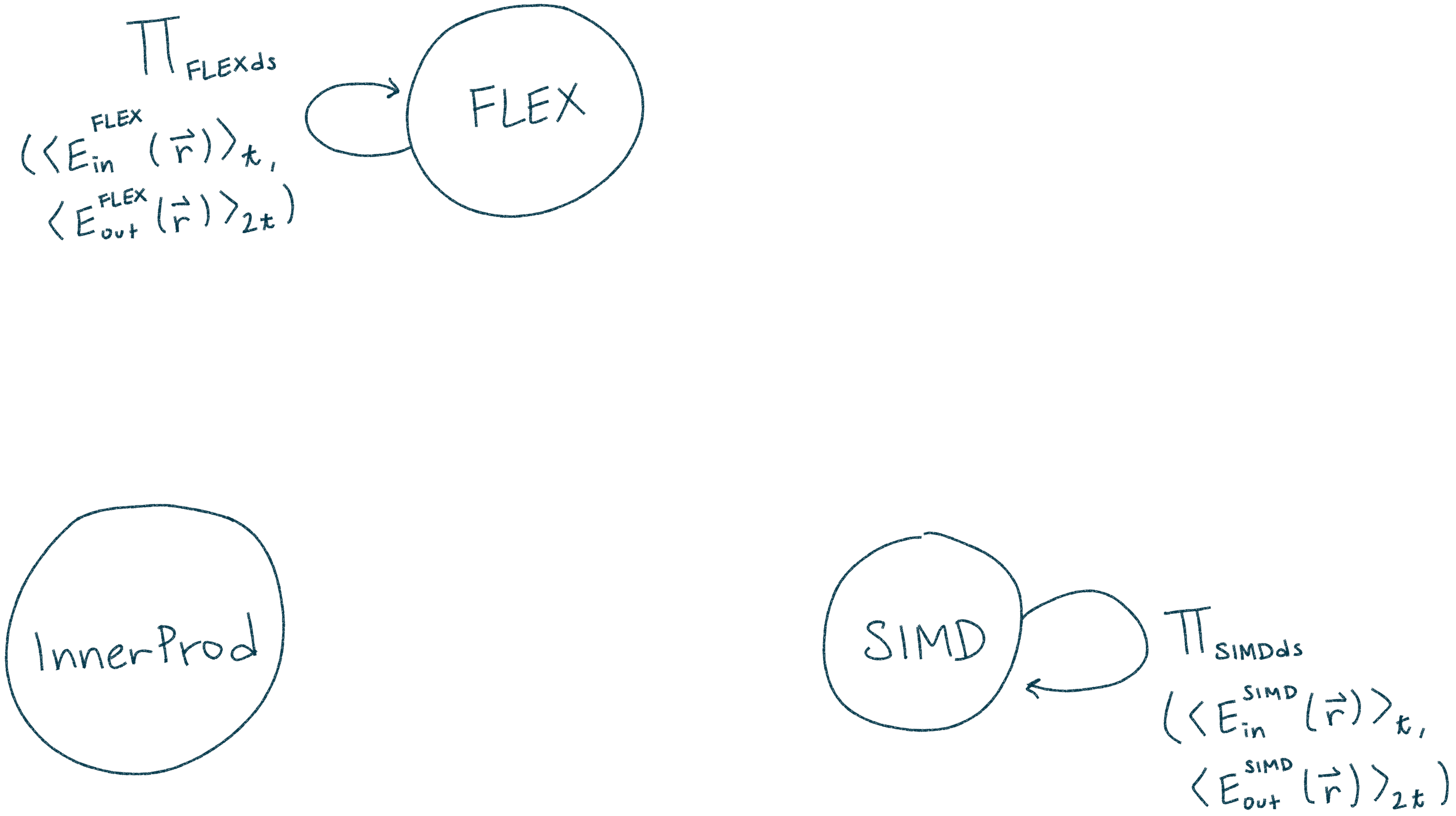
Expressiveness of our encodings

Assuming a single "opening" in $R = GR(2^k, d)$:

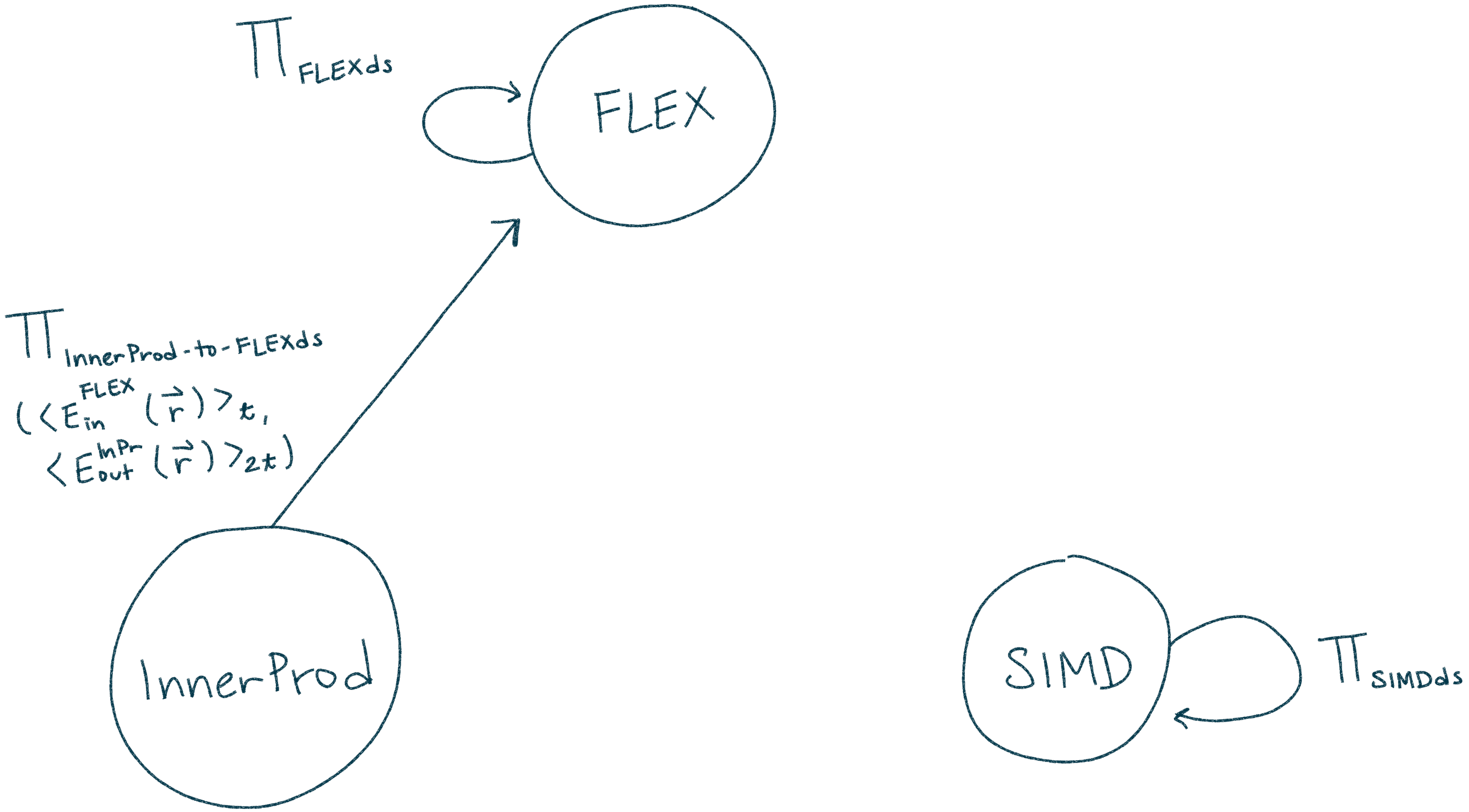
- On 2 inputs:
  - [ACDEY19]: circuits with 1 multiplication and 1 output
  - InnerProd: inner products of length $\approx d/2$
  - SIMD: $\approx d^{0.6}$ parallel circuits with 1 multiplication and 2 output each
- On m inputs:
  - [ACDEY19]: depth 1 circuits with m multiplications and 1 output
  - FLEX: depth 1 circuits with m multiplications and d outputs

[ACDEY 19] "Efficient Information-Theoretic Secure Multiparty Computation over $\mathbb{Z}_{p^k}$ via Galois Rings"
Mark Abspoel, Ronald Cramer, Ivan Damgård, Daniel Escudero, and Chen Yuan

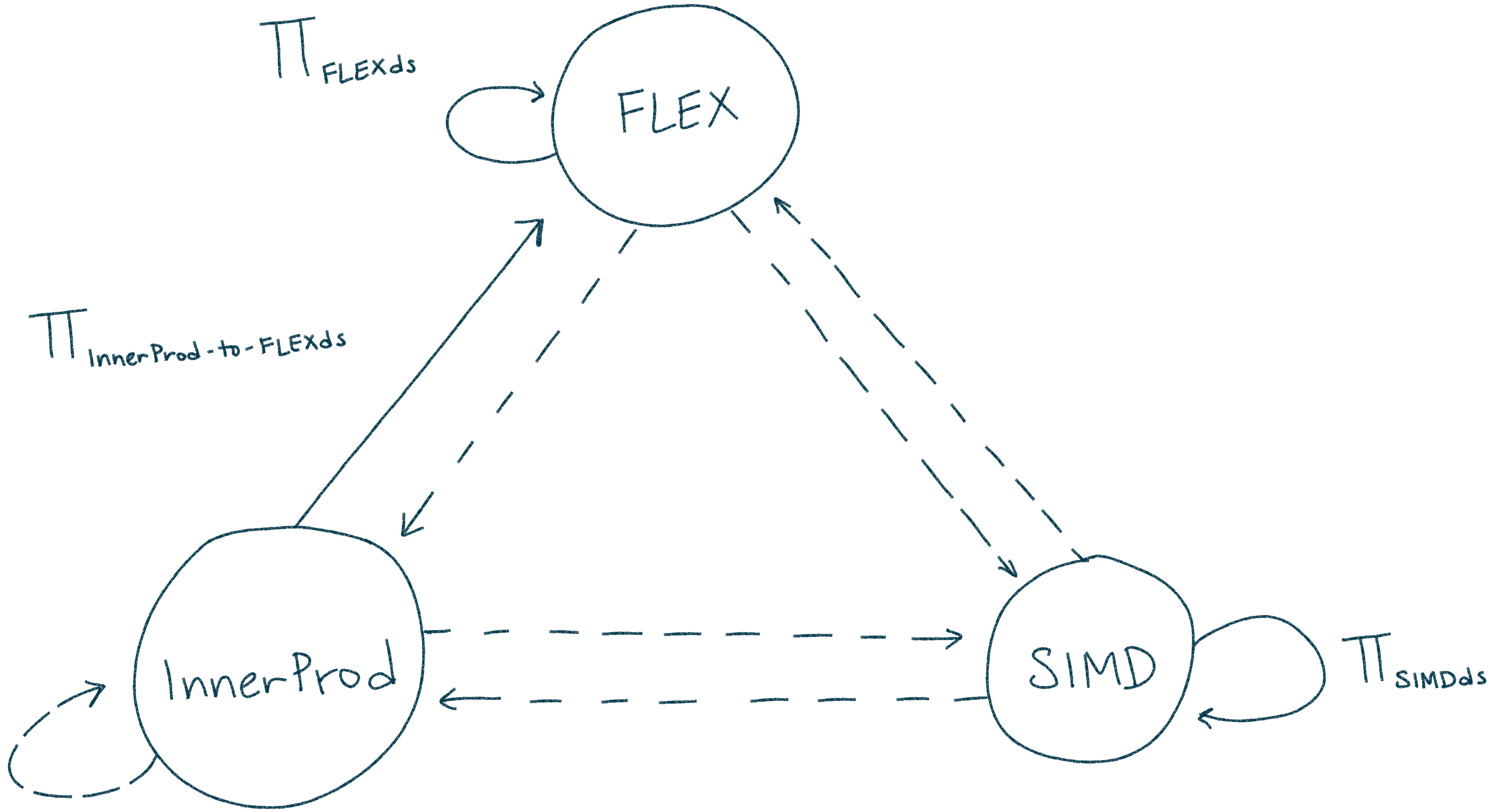
Double shares: Degree reduction + Encoding



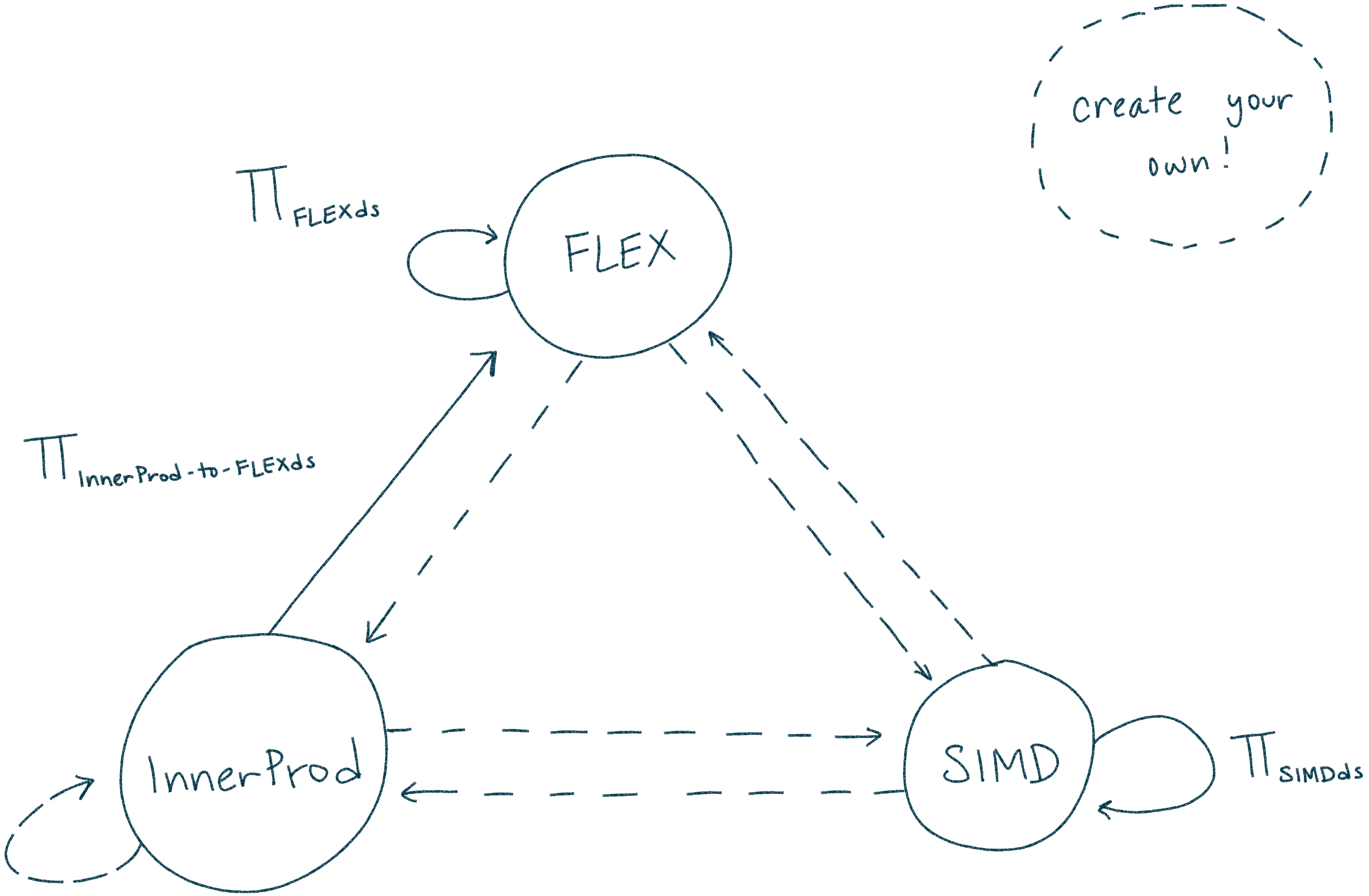
Changing encodings: Double shares



Changing encodings: Double shares



Changing encodings: Double shares



Switching between encodings in $GR(2^k, d)$ and $\mathbb{F}_{2^d}$: daBits

<u>Lemma</u>: Let $\tilde{k} < k$ and $\pi_{\tilde{k}} : GR(2^k, d) \to GR(2^{\tilde{k}}, d)$ be the "reduction mod $2^{\tilde{k}}$" map. Then, $\forall a \in GR(2^k, d)$:

$$\pi_{\tilde{k}}(\langle a \rangle) = \langle \pi_{\tilde{k}}(a) \rangle$$

Where $\pi_{\tilde{k}}(\langle a \rangle)$ is locally computed by parties applying $\pi_{\tilde{k}}$ to their shares of a

<u>Corollary</u>: Let $b \in \{0,1\}$ shared as $\langle b \rangle \in R$. Then $\pi_1(\langle b \rangle) = \langle \pi_1(b) \rangle \in \mathbb{F}_{2^d}$.

sharing in R sharing in $\mathbb{F}$

We obtain daBits [RW19] $(\langle b \rangle^R, \langle b \rangle^{\mathbb{F}})$ at the cost of random bits in $R = GR(2^k, d)$, which allows us to switch between values in R and their bit decomposition (using the same encoding) in $\mathbb{F}_{2^d}$.

[RW 19] "MArBled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security"
Dragos Rotaru and Tim Wood

Second contribution:
Improved double-share production

Producing double-shares: Basic technique

n random values

guaranteed $n-t$ are random

$$\begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix} = \begin{bmatrix} M \\ \text{(Hyper-Invertible)} \end{bmatrix} \begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix}$$

every non-trivial square sub-matrix is invertible

Producing double-shares: Basic technique

guaranteed $n-t$ are random

$$\begin{bmatrix}\langle r_1\rangle_t \\ \vdots \\ \langle r_n\rangle_t\end{bmatrix} = \begin{bmatrix} M \\ (\text{Hyper-Invertible}) \end{bmatrix}\begin{bmatrix}\langle s_1\rangle_t \\ \vdots \\ \langle s_n\rangle_t\end{bmatrix}$$

n random values

Producing double-shares: Basic technique

$$\begin{bmatrix} \langle r_1 \rangle_{2t} \\ \vdots \\ \langle r_n \rangle_{2t} \end{bmatrix} = \begin{bmatrix} M \\ (\text{Hyper-Invertible}) \end{bmatrix} \begin{bmatrix} \langle s_1 \rangle_{2t} \\ \vdots \\ \langle s_n \rangle_{2t} \end{bmatrix}$$

Producing double-shares: Basic technique

t elements (Galois Ring)

$t \cdot d$ elements ($\mathbb{Z}_{2^k}$) [ACDEY 19]

$$\begin{bmatrix}\langle r_1\rangle \\ \vdots \\ \langle r_t\rangle \\ \langle r_{t+1}\rangle \\ \vdots \\ \langle r_n\rangle\end{bmatrix} = \begin{bmatrix} M \\ \text{(Hyper-Invertible)} \end{bmatrix} \begin{bmatrix}\langle s_1\rangle \\ \vdots \\ \langle s_n\rangle\end{bmatrix}$$

Our solution: Batch check for double-shares

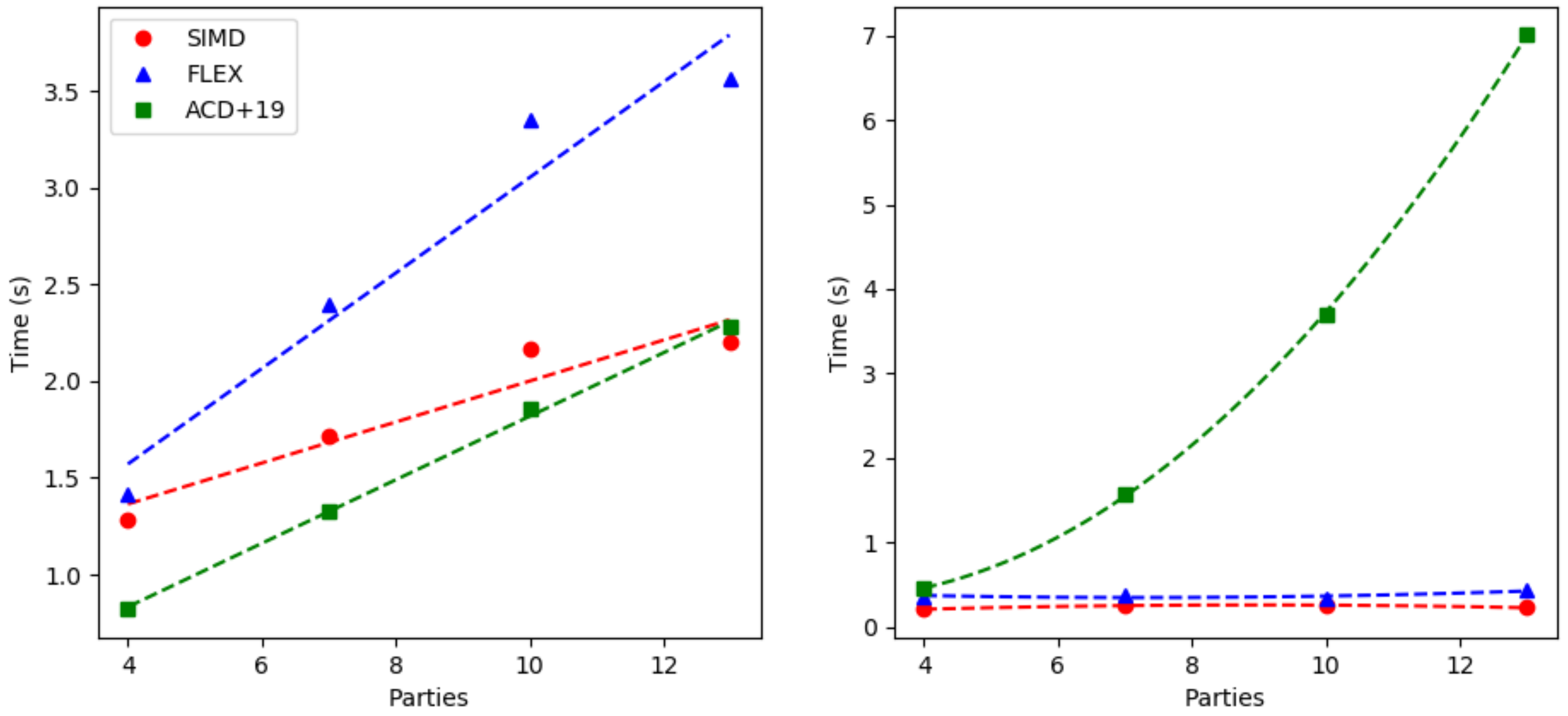
$$\text{2t encodings}\left\{\begin{bmatrix}\langle E(r_1)\rangle\\ \vdots \\ \langle E(r_{2t})\rangle\end{bmatrix}\right. = \begin{bmatrix} M \\ \text{(Hyper-Invertible)}\end{bmatrix}\begin{bmatrix}\langle E(s_1)\rangle\\ \vdots\\ \langle E(s_n)\rangle\end{bmatrix}$$

Batch check of resulting $E(r)$ values.

Challenges:
* Action of $M \in \mathcal{M}_{n\times n}(GR(2^k, d))$ has to preserve encodings
* zero divisors — careful with random linear combinations

Experimental results

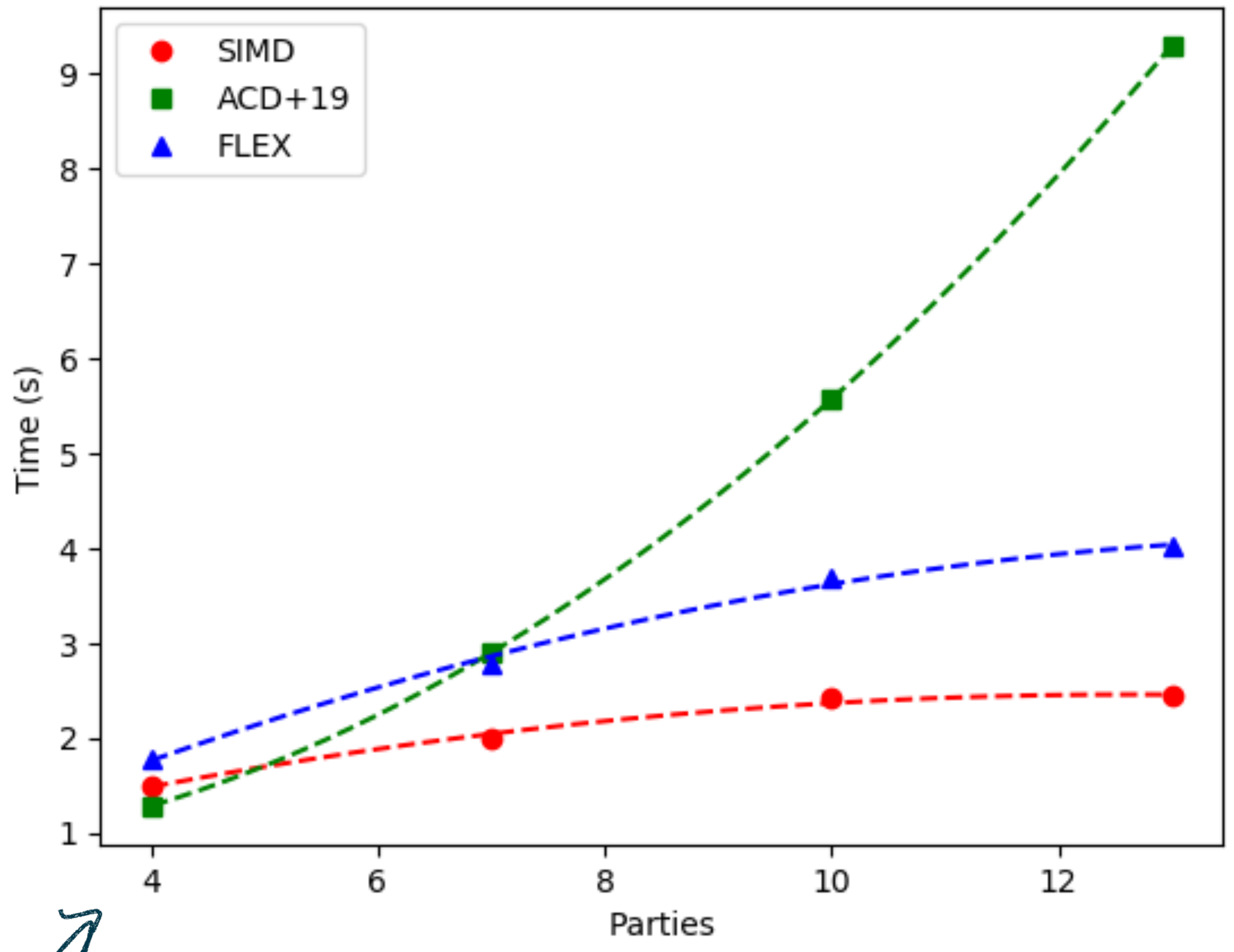
Running time for generation (left) and check (right) step of double share protocols for 126k double-shares



small overhead in generating encoded double shares

Batch check shows noticable improvement

Combined running time of generating and checking 126k double-shares



combined, our method has higher throughput for larger n

Contributions recap:

1. Encodings for $GR(2^k, d)$: Exploiting d to encode circuits of $\mathbb{Z}_{2^k}$.
    - Just set $k = 1$ to use $\mathbb{F}_{2^d}$ to encode circuits over $\mathbb{F}_2$
    - Framework to construct other encodings and “translate” between them
2. Batch checks for (encoded) double-shares
    - Faster preprocessing for [BH08]-style protocols (stat. security)
3. Random bits in $GR(2^k, d) \Rightarrow$ daBits from $GR(2^k, d)$ to $\mathbb{F}_{2^d}$
    - Improved preprocessing for conversions between linear secret sharing schemes over $\mathbb{Z}_{2^k}$ and $\mathbb{F}_{2^d}$

Contributions recap:

1. Encodings for $GR(2^k, d)$: Exploiting d to encode circuits of $\mathbb{Z}_{2^k}$.
    - Just set $k = 1$ to use $\mathbb{F}_{2^d}$ to encode circuits over $\mathbb{F}_2$
    - Framework to construct other encodings and "translate" between them
2. Batch checks for (encoded) double-shares
    - Faster preprocessing for [BH08]-style protocols (stat. security)
3. Random bits in $GR(2^k, d)$ $\Rightarrow$ daBits from $GR(2^k, d)$ to $\mathbb{F}_{2^d}$
    - Improved preprocessing for conversions between linear secret sharing schemes over $\mathbb{Z}_{2^k}$ and $\mathbb{F}_{2^d}$