The Picnic Post-Quantum Signature Scheme and its Security Analysis

Itai Dinur

Ben-Gurion University

Credit for some slides: Melissa Chase and Picnic team

Post-Quantum Cryptography

- Large-scale quantum computer could efficiently **factor** large numbers and **compute discrete logs**
  - **Breaks hardness assumptions** of all standardized public key crypto (e.g., RSA, DSA, ECDSA)
- Goal of **post-quantum crypto**: design **new schemes** that:
  - can be run on **classical** computer
  - remain secure even if adversary has a **quantum computer**

Post-Quantum Crypto Standardization

- NIST (National Institute of Standards and Technology) initiated post-quantum crypto **standardization project**
  - Goal: **standardize** post-quantum crypto schemes **by 2024**
  - Submission deadline: November 2017 (69 accepted)
- Why now? existing quantum computers extremely limited
  - Some researchers believe that a fundamental public-key crypto scheme may be **broken** by a quantum computer **by 2030**
  - Designing and deploying (secure) cryptography is **slow**

Post-Quantum Crypto Standardization

- Scope:
  - Digital signatures
  - Public-key encryption
  - Key-establishment
- Main selection criteria
  - **Security** against both **classical** and **quantum** attacks
  - **Performance** on various "classical" platforms

Post-Quantum Crypto Design

- Factoring and discrete log are **not hard problems** on a quantum computer

- (Conjectured) hard problems:
    - Problems on **algebraic structures** (lattices, codes, Multi-variate polynomials…)
    - **Symmetric-key algorithms** (hash functions, block ciphers, pseudo-random generators)

Signatures from Symmetric-Key Algorithms

- In this talk: focus on **signature schemes**
- Can be built using **symmetric-key algorithms:**
- **Hash-based signatures** based on Lamport’s **one-time signatures** (1979)
- Practical challenge: efficiency (+compatibility)
- A lot of **progress** in recent years

Picnic

- New **signature scheme** based on **symmetric-key algorithms**
  - Submitted to NIST's project
- Built **completely differently** from hash-based signatures
- New design: a lot of **room for optimizations**

	Public key size	**Signature size**	**Signing time**	**Verification time**	**Post-quantum security**
ECDSA	Small	Small	Fast	Fast	-
Picnic	Small (100's bits)	Moderate (10K's bits)	Moderate (ms's)	Moderate (ms's)	+

Picnic Designers

PICNIC was designed by a group of cryptographers from Aarhus University, AIT Austrian Institute of Technology GmbH, DFINITY, Graz University of Technology, Georgia Tech, Microsoft Research, Northwestern University, Princeton University, Technical University of Denmark and the University of Maryland. The team includes Melissa Chase, David Derler, Steven Goldfeder, Jonathan Katz, Vladimir Kolesnikov, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Xiao Wang, and Greg Zaverucha.

In this Talk

- **Basic design of Picnic**
- Optimizations
- Security Analysis

Digital Signature Scheme

- A digital signature scheme defines 3 algorithms:
- **Key generation algorithm** (run by **signer**) outputs:
  - SK (secret signing key)
  - PK (public verification key)
- **Signing algorithm** (run by **signer**):
  - Inputs: SK,m
  - Output: signature s
- **Verification algorithm** (run by **verifier**):
  - Inputs: PK,m,s
  - Output: signature s on m "**valid**" or "**not valid**"

Picnic Signature Scheme: Overview

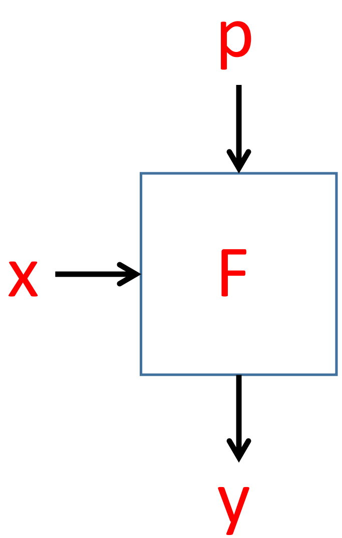
- PK = F(SK) for some function F
- F must be **hard to invert** (not leak SK)

- A signature is **proof of knowledge** of SK (with m as nonce)
- Proof (=signature) must not leak SK, so must be a **zero knowledge (ZK) proof**

- Require:
  - Hard to invert function F
  - ZK proof system

Picnic Signature Scheme: Overview

- F is implemented using a **block cipher**
- **Key generation algorithm:**
  - Choose random **plaintext block** p and **key** x for F and compute y=F(x,p) (encrypt p using key x)
  - SK=(p,x)
  - PK=(p,y)



- sign((p,x),m)
  - Output s = proof of knowledge of x such that y=F(x,p) (with m as nonce)

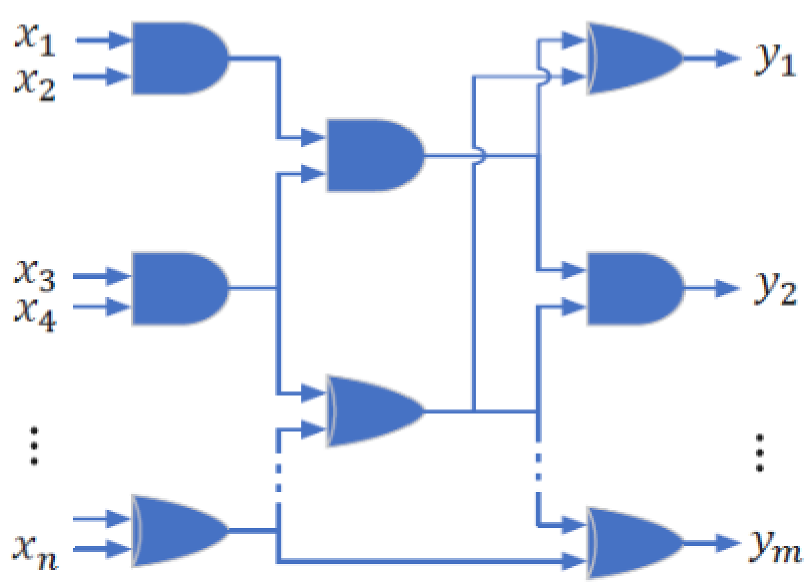
Picnic's Zero Knowledge Proof

- Prove knowledge of x such that y=F(x,p) (with m as nonce)
- Represent F as a **Boolean circuit** C, with output $y=y_1,y_2,\ldots,y_m$
- Prove knowledge of $x=x_1,x_2,\ldots,x_n$ such that y=C(x)
  - Note: p is fixed ("hardwired to C")
- Signer proves in ZK "I know x such that C(x)=y"

SK (private)

F,p (pubic)

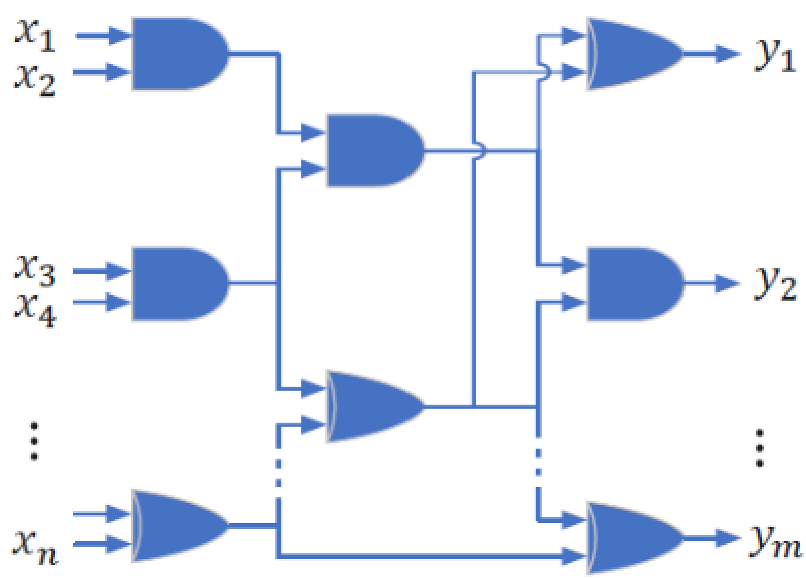
PK (pubic)



Picnic's Zero Knowledge Proof

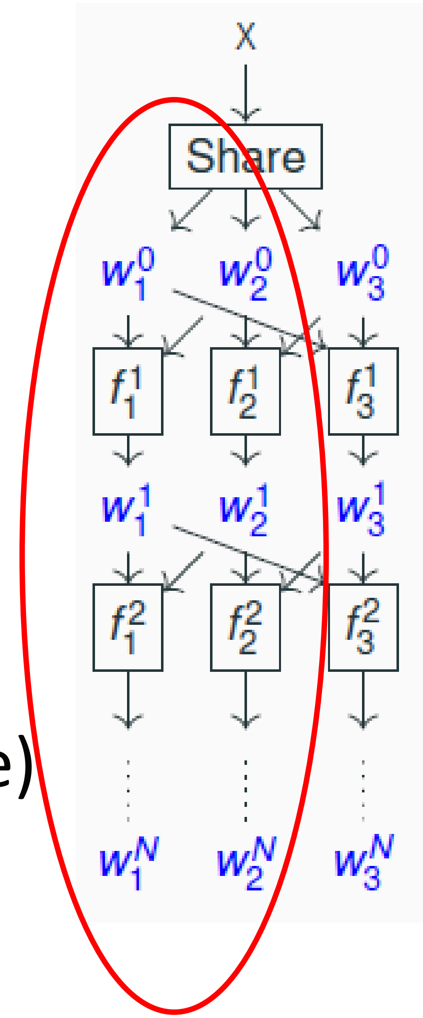
- Building blocks:
  - Multi-Party Computation (MPC-in-the-Head [IKOS07])
  - Commitment scheme

SK (private) F,p (pubic) PK (pubic)



MPC (Multi-Party Computation)

- (Special) MPC Setting:
  - **Public** Boolean circuit C, **secret** input value x
  - t players, player i given **input share** w_i^0
  - $w_1^0 \oplus w_2^0 \oplus \ldots \oplus w_t^0 = x$
  - **Goal**: compute **output shares** $w_1^N, w_2^N, \ldots, w_t^N$
  - $w_1^N \oplus w_2^N \oplus \ldots \oplus w_t^N = C(x)$
- Players communicate
- **Privacy requirement**:
  - if t-1 players combine information, **learn nothing** about x (or missing player's share)



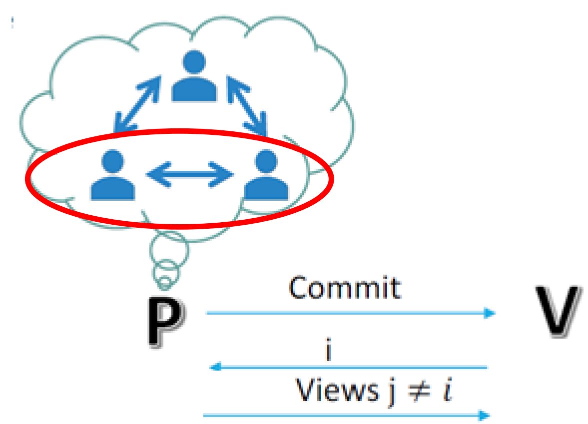
Hash-Based Commitment Scheme

- **Committing** to a value v
  - Choose random string k
  - Output **commitment**: z=H(v,k) for crypto hash function H
- **Opening** a commitment
  - Reveal v,k
  - Given z and v,k, anyone can verify that z=H(v,k)

- **Hiding property**: commitment z **hides** v
- **Security property**: Given commitment z to value v, committer "**cannot lie**" about v

ZK from MPC: MPC-in-the-Head [IKOS07]

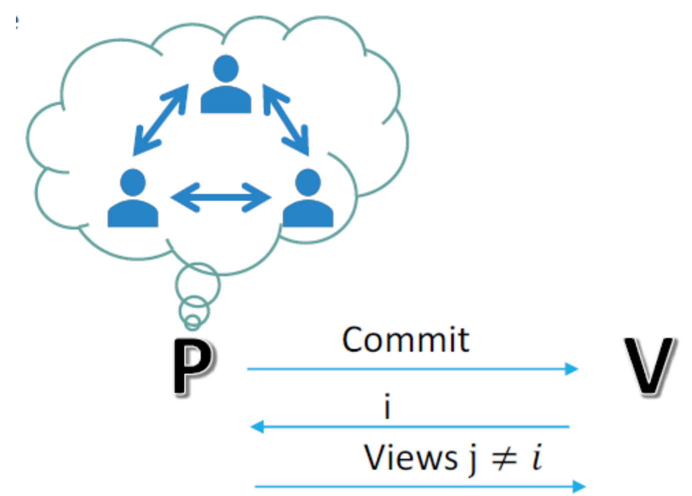
- In Picnic, signer proves "I know x such that C(x)=y"
- Assume signing\verification is an **interactive** process:
  - Prover chooses t=3 random shares s.t. $w_1^0 \oplus w_2^0 \oplus w_3^0 = x$
  - Imagine t=3 parties each with input w_i^0
  - **Internally** run MPC to compute w_1^N, w_2^N, w_3^N s.t. $w_1^N \oplus w_2^N \oplus w_3^N$=C(x)=y
  - For each player, **commit** to "**view**":
    - input w_i^0, randomness, states, messages sent and received
  - Verifier chooses random **challenge** $i \in \{1,2,3\}$
  - Prover **reveals** views of 2 players except i
  - Verifier checks:
    - **(Partial) correctness** of MPC computation
    - **Openings** of 2 commitments



MPC-in-the-Head [IKOS07]

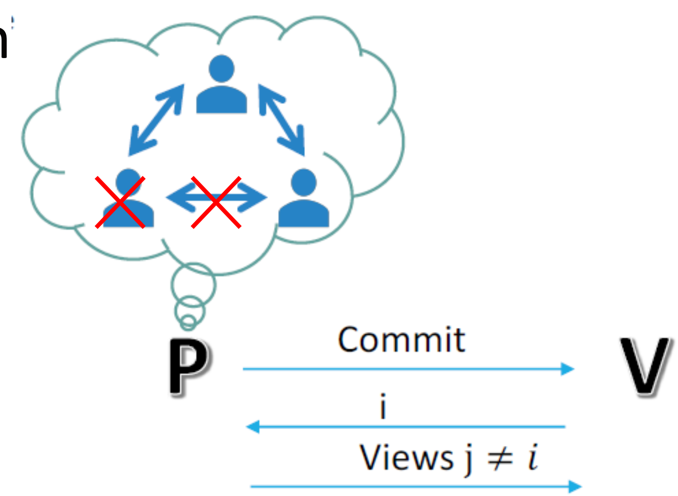
- **Zero Knowledge**: Verifier **learns nothing** about x by privacy of MPC protocol (sees only 2 out of 3 views)

- **Correctness**: If prover knows x, can run MPC protocol correctly and **pass verification**



MPC-in-the-Head [IKOS07]

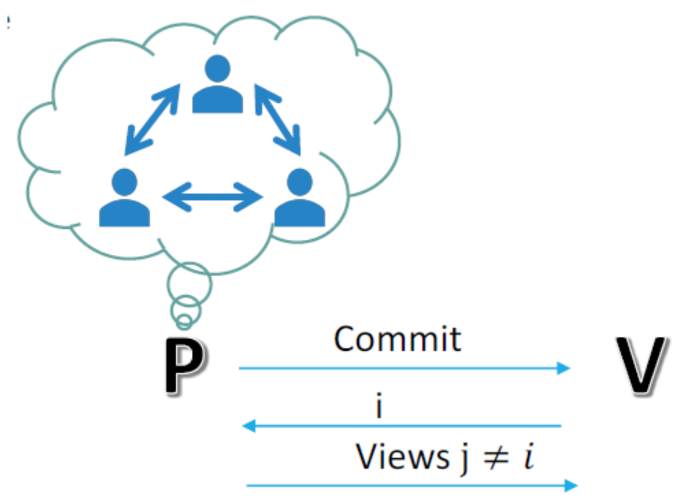
- **Soundness** (proof convincing?):
- If prover doesn't know x and tries to **cheat**, either:
  - A player misbehaved
  - 2 views are inconsistent
- **Catch** cheater with probability p=1/3
- Repeat R **times** to amplify p
  - R=219 times for $p = 1-(2/3)^{219} \approx 1-2^{-128}$ (128-bit security)
- Why simulate 3 players?
  - 2 players give soundness 0 (cannot check consistency)
  - 4 players: **better soundness** 2/4 per run but much **more communication**
  - In general: **all pairs** communicate. Communication increases **quadratically**
  - More communication = larger proof = larger signature, signing time



Removing Interaction

- Problem: signing\verification is not **interactive**
- How to generate R ``random'' challenges $i_r \in \{1,2,3\}$?
- Solution: in sign((p,x),m) use **Fiat-Shamir transform**
- Generate challenges as H(commitments,m)
  - Challenges pseudorandom and cannot be predicted

- **Signature** s includes for each **run** r=1,2,...,R:
  - 3 commitments, views of 2 players except i_r

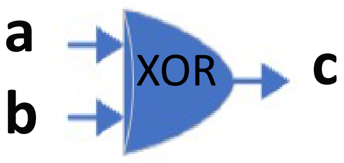


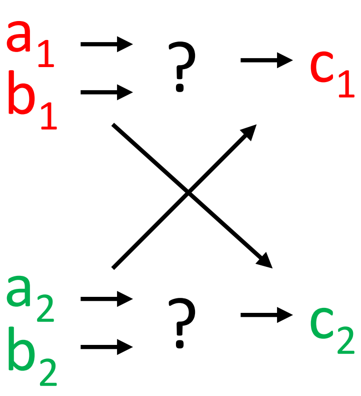
In this Talk

- Basic design of Picnic
- **Optimizations**
- Security Analysis

Picnic's MPC Protocol (ZKBoo [GMO16])

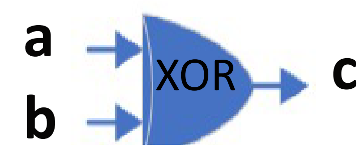
- For each wire with Boolean value **a** in C: each player 1,2,3 holds wire with (resp.) Boolean value a_1,a_2,a_3
- **Invariant**: for each wire with value **a**, $a_1 \oplus a_2 \oplus a_3 = \mathbf{a}$
- Assume 2 players, XOR gate $\mathbf{a} \oplus \mathbf{b} = \mathbf{c}$
- Know that $a_1 \oplus a_2 = \mathbf{a}$, $b_1 \oplus b_2 = b$
- Need to define c_1,c_2 such that $c_1 \oplus c_2 = \mathbf{c}$
    - Players don't learn information



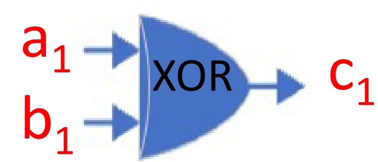


Picnic's MPC Protocol (ZKBoo [GMO16])

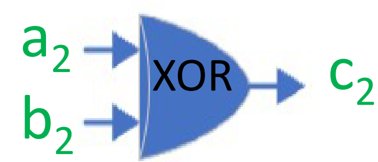
- For each wire with Boolean value **a** in C: each player 1,2,3 holds wire with (resp.) Boolean value a_1, a_2, a_3
- **Invariant**: for each wire with value **a**, $a_1 \oplus a_2 \oplus a_3 = \mathbf{a}$
- Assume 2 players, XOR gate $\mathbf{a} \oplus \mathbf{b} = \mathbf{c}$
- Know that $a_1 \oplus a_2 = \mathbf{a}$, $b_1 \oplus b_2 = b$
- Need to define c_1, c_2 such that $c_1 \oplus c_2 = \mathbf{c}$
  - Players don't learn information



- Define: $c_1 = a_1 \oplus b_1$, $c_2 = a_2 \oplus b_2$
- $c_1 \oplus c_2 = (a_1 \oplus b_1) \oplus (a_2 \oplus b_2) =$
  $(a_1 \oplus a_2) \oplus (b_1 \oplus b_2) = \mathbf{a} \oplus \mathbf{b} = \mathbf{c}$

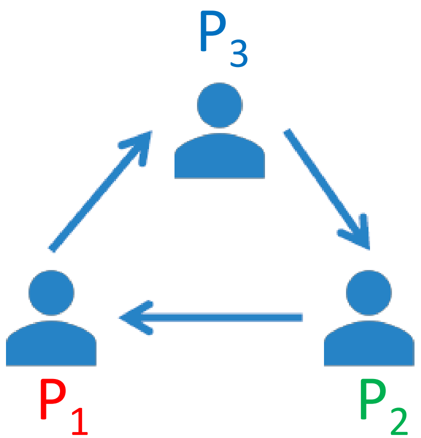


- **XOR** computation is **local:** No need to include XOR outputs c_1, c_2 in signature
  - Verifier **computes outputs** of **XOR** gates from known inputs



Picnic's MPC Protocol (ZKBoo [GMO16])

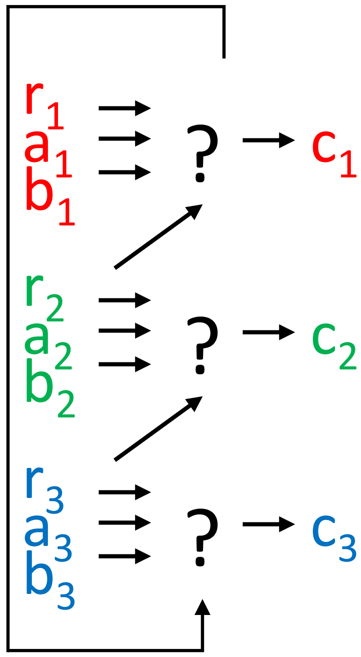
- Maintaining invariant for AND gates is more **complicated**
- Requires parties to **communicate**, generate **random bits**
- "MPC-in-the-head" optimizations:
  - Player P_i only depends on player P_{i+1}
  - Instead of sending messages: define current state of P_i as **function** of previous state, current state of P_{i+1}
  - Given (open) states of P_i , P_{i+1}, **consistency** can be checked by verifier – no "messages" in signature (proof)



Picnic's MPC Protocol (ZKBoo [GMO16])

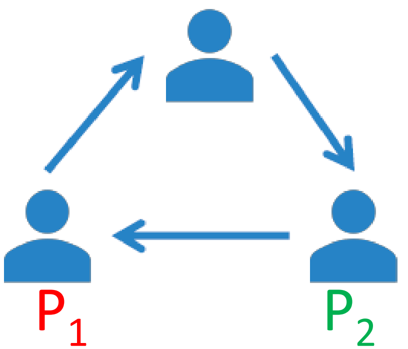
- AND gate implementation **c=a·b**
- Parties generate **random bits** r_1, r_2, r_3

$$c_1 = a_1 \cdot b_1 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus r_1 \oplus r_2$$
$$c_2 = a_2 \cdot b_2 \oplus a_3 \cdot b_2 \oplus a_2 \cdot b_3 \oplus r_2 \oplus r_3$$
$$c_3 = a_3 \cdot b_3 \oplus a_1 \cdot b_3 \oplus a_3 \cdot b_1 \oplus r_3 \oplus r_1$$



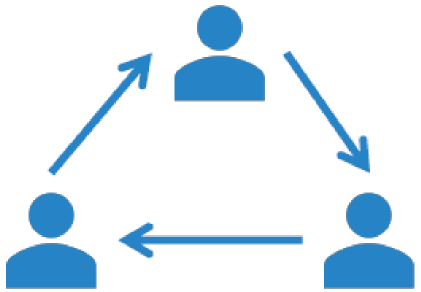
- Assume views of P_1, P_2 opened
- Verifier checks consistency:

$$c_1 = a_1 \cdot b_1 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus r_1 \oplus r_2$$



Picnic's MPC Protocol (ZKBoo, ZKB++)

- **XOR** gates do not blow up signature, **cheap** to compute
- **AND** gates blow up signature size (randomness, additional state), more **expensive** to compute
- Optimizations:
  - **Circuit** C: Use (secure) **block cipher** with **small** number of **AND** gates – LowMC [ARS+15]
  - **Randomness** generation: each player generates (pseudo) random bits deterministically using **PRG** from **short random seed**
  - View of each open player (in signature) includes **short seed**
    - Instead of random bits



In this Talk

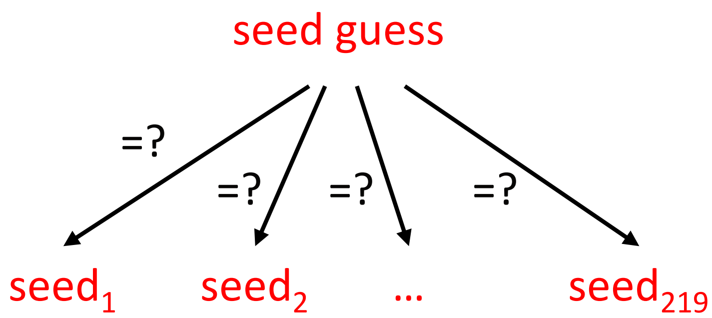
- Basic design of Picnic
- Optimizations
- **Security Analysis**

Security Analysis ([D,Nadler 2018])

- Consider Picnic variant for 128-bit security
- Attacker given **signature** with R=219 partial MPC **runs**
- Each partial run r exposes 2 out of 3 player views
  - Includes 2 random 128-bit seeds
- 3'rd seed unexposed – if revealed allows to **easily compute** block cipher (signing) key
- Attack attempt: given run, **guess** unknown 128-bit seed
  - Complexity: 2^{128}

Security Analysis ([D,Nadler 2018])

- **Multi-target** attack:
  - Given signature, store **all** R=219 runs
  - **Guess** unopened player's seed for **one** of 219 runs (targets)
  - Complexity: $\frac{2^{128}}{219} \approx 2^{120}$



- Problem: how to **detect** seed guess = seed_r?

- Seems impossible: MPC **protects** unopened player privacy

Security Analysis ([D,Nadler 2018])

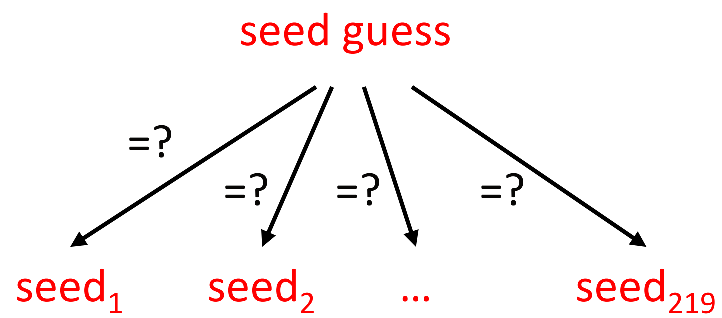
- **Subtlety**: MPC protects player's **input**, but not generated **random bits**

$$c_1 = a_1 \cdot b_1 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus r_1 \oplus r_2$$
$$c_2 = \cancel{a_2 \cdot b_2} \oplus \cancel{a_3 \cdot b_2} \oplus \cancel{a_2 \cdot b_3} \oplus r_2 \oplus \mathbf{r_3}$$
$$c_3 = a_3 \cdot b_3 \oplus a_1 \cdot b_3 \oplus a_3 \cdot b_1 \oplus r_3 \oplus r_1$$

- Assume P_1,P_2 opened. Goal: determine $\mathbf{r_3}$
- Assume $a_2=b_2=0$
- $\mathbf{r_3}=c_2 \oplus r_2$

Security Analysis ([D,Nadler 2018])

- **Multi-target** attack:
  - Complexity: $\frac{2^{128}}{219} \approx 2^{120}$



- Problem: how to **detect** seed guess = seed_r?
  - For each run: compute PRG bits produced by unopen player (PRG(seed_r)), sort in table
  - Compute PRG(seed guess), search in table
- In practice attack **more complex**
  - For each run can compute **different PRG output bits** for unopened player
  - Simple sort-and-match doesn't work

Security Analysis ([D,Nadler 2018])

- Generalization: given S signatures with $S{\cdot}219$ runs
  - Signed by **one** or **many** users
  - Attack complexity: $\frac{2^{128}}{S\cdot 219} \approx \frac{2^{120}}{S}$
  - E.g. given 2^{45} signatures, security **reduced** from 2^{128} to 2^{75}
- Weakness exists in several related cryptosystems
- Fix (Picnic 2.0): **salt** PRG
  - Player i in run r produces random bits using PRG($salt_{i,r}$, $seed_{i,r}$)
  - Forces attacker to **choose salt** when evaluating PRG
  - Can only compare with 1 target

Conclusions

- Picnic is a new **promising** post-quantum signature scheme
  - A lot of room for **improvements**
- Optimizes (traditionally) theoretical crypto for practical use
  - **Requires care:** consider "real world" attacks

Thanks for your attention!