Sabre: A speedier and scalable Riposte

Adithya Vadapalli
joint work with
Kyle Storrier and Ryan Henry

Indiana University and University of Calgary

21st August 2020
University of Calgary

Riposte, Oakland 2017

Goal

The goal of riposte is to do anonymous broadcasting.

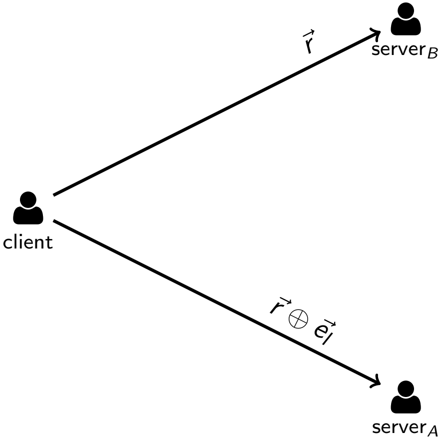
A Simple Construction

Goal

1. Client wants to write $\mathbf{1}$ into row l of the database.
2. Servers hold shares of an L-bit string (a database with 1-bit messages)

1. Client generates a random string r (length L) and sends it to A.
2. Client sends to $r \oplus e_l$ to B.
3. The servers XORs, the received string with its share of the database.

Riposte, Oakland 2017



Riposte, Oakland 2017

Problem

The main problem with the simple approach is the *communication* cost.

DPFs

A rough one-line definition of DPFs. They are a way to share a standard basis vector among two parties by sending them short PRG seeds.

Distributed Point Function, CCS 2016, Eurocrypt 2014

Definition

The *point function* at I over $\mathbf{GF}(2^\lambda)$ is the function $P\colon \mathbf{GF}(2^\lambda) \to \mathbf{GF}(2^\lambda)$ defined via

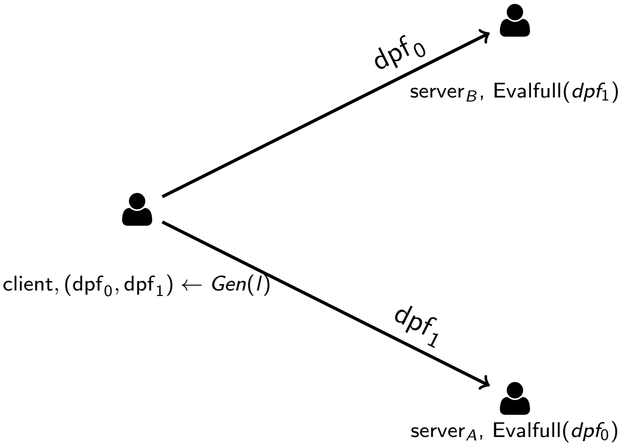
$$P(j) = \begin{cases} \mathbf{1} & \text{if } j = I \text{, and} \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

Definition

A distributed point function (DPF) is a pair of PPT algorithms $\mathrm{DPF} = (\mathrm{Gen}, \mathrm{Eval})$ where:

- **Gen**(x), with $x \in \{0,1\}^*$, outputs a pair of keys $(\mathrm{dpf}_0, \mathrm{dpf}_1)$.
- **Eval**(k, x') with $k, x' \in \{0,1\}^*$, such that **Eval**$(\mathrm{dpf}_0, x') \oplus$ **Eval**$(\mathrm{dpf}_1, x') = 1$ if $x' = x$, otherwise 0.
- **Evalfull** evaluates the point function over the entire range.
- **Evalfull**$(\mathrm{dpf}_0) \oplus$ **Evalfull**$(\mathrm{dpf}_1) = \vec{e}_I$.

Riposte



dpf_0

server$_B$, Evalfull(dpf_1)

client, $(\mathrm{dpf}_0, \mathrm{dpf}_1) \leftarrow Gen(I)$

dpf_1

server$_A$, Evalfull(dpf_0)

1. Client generates DPF keys and sends it to the servers.
2. Recall $\mathrm{Evalfull}(\mathrm{dpf}_0) \oplus \mathrm{Evalfull}(\mathrm{dpf}_1) = \vec{e}_I$
3. Servers compute Evalfull(dpf_0) and Evalfull(dpf_1). Then they XOR them to their share of the database.

Malicious Clients

Malicious clients can send bogus DPF seeds and corrupt the database.

Protect against malicious clients

The two servers need to verify that DPF seeds are well-formed.

Zero Knowledge Proofs

Riposte use ZKPs to ensure that the DPFs are well-formed.

What are Zero-Knowledge Proofs?

Definition

The prover wants to prove the knowledge of a statement to the verifier. The goal is to prove knowledge of the statement, with the verifier learning *nothing* else.

Slightly more formally,

1. Let L be a language in NP and let $R(x, w)$ be the corresponding NP-relation. (x is the public input, w is the witness).
2. Prover proves the "knowledge" w, without revealing w itslef.

ZKPs for Riposte

Goal

1. The client which generates dpf keys dpf_0 and dpf_1.
2. Wants to convince the servers that, $\text{Evalfull}(\text{dpf}_0) \oplus \text{Evalfull}(\text{dpf}_1)$ is a standard basis vector.
3. Cannot reveal DPF keys dpf_0 and dpf_1.

Less efficient DPFs

Riposte uses $O(\sqrt{n})$ sized-DPFs; while the most efficient DPFs are of size $O(logn)$.

Our Contribution

Sabre uses the most efficient, $O(logn)$-sized DPFs.

MPC in the head, STOC 2007

Multi-Party Computation and Zero-Knowledge Proofs

MPC in the head is a paradigm that uses MPC to do ZKP.

What is Multi-Party Computation?

Definition

Parties $\mathrm{P}_1, \cdots, \mathrm{P}_n$ have private inputs $w_1, w_2, \cdots, w_n$ respectively. They run a protocol among themselves to compute a function $f(w_1, \cdots, w_n)$.

t-privacy

The protocol is secure against a coaltion of atmost t corrupt participants.

MPC in the head, STOC 07

Prover

1. $f(x, w_1, w_2, \cdots, w_n) = R(x, w_1 \oplus \cdots \oplus w_n)$, where $(w_1 \oplus \cdots \oplus w_n = w)$
2. Prover simulates an MPC protocol **in their head** to compute $f(\mathrm{w}_1, \cdots, \mathrm{w}_n)$.
3. Prover commits to the transcript of the simulated 2-private MPC protocol.

MPC in the head, STOC 07

Verifier

1. Verifier selects 2 parties at random and asks the verifier to reveal the transcript.
2. Verifier checks that:
   - 2.1 The transcripts are consistent with each other.
   - 2.2 The output is correct.

MPC in the head, STOC 07

Soundness

Soundness error $= 1/\binom{n}{2}$

Soundness

Error probability can be reduced to 2^{-k} by repeating the experiment $O(kn^2)$ times.

Coming back to Sabre

Recall

The client wants to prove that dpf_0 and dpf_1 are valid DPF keys.

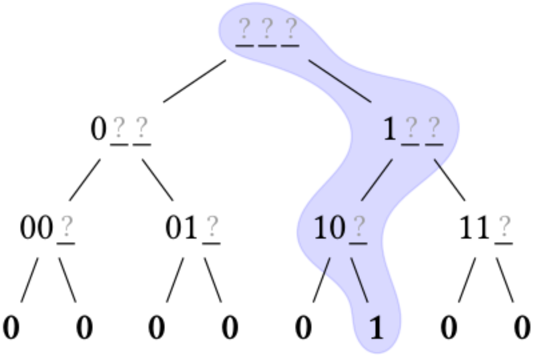
MPC

1. The client first creates shares of the keys, dpf_0 and dpf_1.
2. Then, it runs an MPC protocol **in her head**.

Point Functions, revisited

The *point function* at i over $\mathbf{GF}(2^\lambda)$ is the function $P\colon \mathbf{GF}(2^\lambda) \to \mathbf{GF}(2^\lambda)$ defined via

$$P(j) = \begin{cases} \mathbf{1} & \text{if } j = i\text{, and} \\ \mathbf{0} & \text{otherwise.} \end{cases}$$



Properties

Type 0 nodes

1. it is a leaf with label “0”
2. it is a non-leaf and both of its children are of type 0;

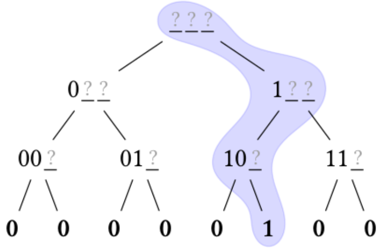
Type 1 nodes

1. it is a leaf with label “1”
2. it is a non-leaf with exactly one type-1 child and one type-0 child.

Observation

If a tree is rooted at a 0-node, then all of its leafs are of type 0. If a tree is rooted at a 1-node, then exactly one of its leafs is of type 1 and all others are of type 0.

Point Functions, revisited



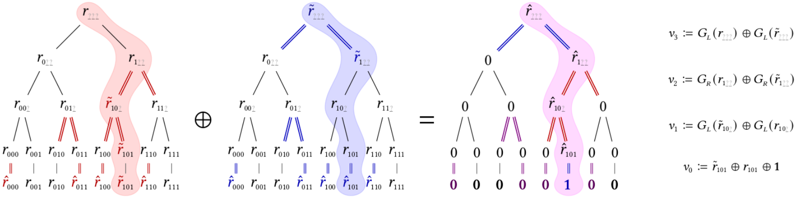
1-path

A path from the root to the leaf comprising of 1-nodes is called a 1-path.

Key Observation

A function is a point function if and only if it has a 1-path.

Distributed Point Function



(a) First tree share (b) Second tree share (c) Reconstructed tree (d) Correction words

$$v_3 := G_L(r_{\underline{???}}) \oplus G_L(\tilde{r}_{\underline{???}})$$

$$v_2 := G_R(r_{1\underline{??}}) \oplus G_R(\tilde{r}_{1\underline{??}})$$

$$v_1 := G_L(\tilde{r}_{10\underline{?}}) \oplus G_L(r_{10\underline{?}})$$

$$v_0 := \tilde{r}_{101} \oplus r_{101} \oplus \mathbf{1}$$

At Every level

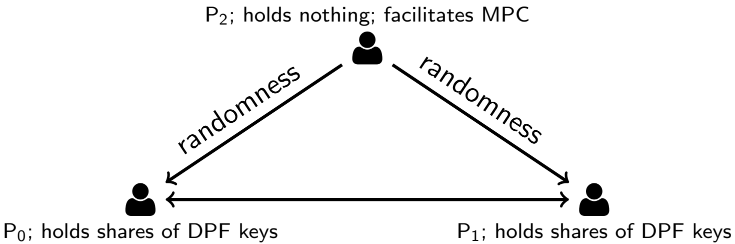
1. P_0 : computes $L_0 \| R_0 = PRG(seed_0) + b \cdot cw$
2. P_1 : computes $L_1 \| R_1 = PRG(seed_1) + b \cdot cw$ $(b \in \{0, 1\})$
3. Either $L_0 = L_1$ or $R_0 = R_1$

MPC for DPFs

Our MPC Protocol

- Proves the existence of a 1-path.
- Evaluates the 1-path.
- Proving the existence of a 1-path is equivalent to showing that every level of DPF computation, *exactly* one half of the PRG evaluation reconstructs to 0.

MPC for DPFs



Things to know about our MPC protocol

1. P_2 uses a PRG seeds to create randomness for P_1 and P_2.
2. P_0 and P_1 receive some randomness and communicate with each other.
3. We use *LowMC* block cipher to implement the PRG.

MPC for DPFs

3 Party MPC

1. To implement our MPC we use 1-private 3-party MPC prtocol.
2. This means that, a single verifier can look at the transcript of at most one party.

Multiple Verifiers

1. We solve this problem by introducing another verifier.
2. We have two versions, nameley 2 Verifier and 3 Verifiers.

2 Verifier MPC-in-the-head

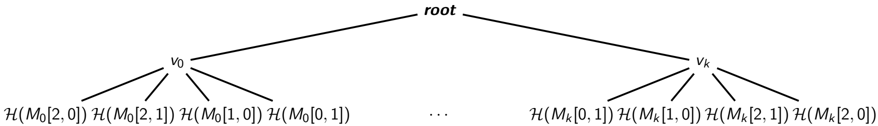
Simulation

1. The simulator (the prover) runs K independent simulations of the MPC protocol.
2. $M_i[x, y]$ ordered set of messages sent from P_x to P_y

Merkle-Tree Construction

Prover

The prover constructs a Merkle-tree by hashing each of the ordered pairs of messages between the parties.



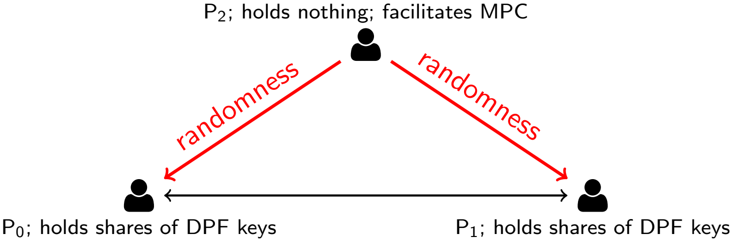
Proof for Verifier 0 (other verifier is symmetrical)

- The root of the Merkle-tree (Let c_i be i^{th} bit of the root).
- For all i, such that $c_i = 1$:
  - $M_i[0,1]$, $M_i[2,0]$.
  - $\mathcal{H}(M_i[2,1])$, $\mathcal{H}(M_i[2,0])$.
- For all i, such that $c_i = 0$:
  - $\mathcal{H}(M_i[0,1])$, $\mathcal{H}(M_i[1,0])$.
  - seed_i; the seed used by P_2 to generate the randomness.

Verifier 0, i^{th} iteration (the other verifier is symmetrical)

Case A, $c_i = 0$, Does P_2 follows the protocol?

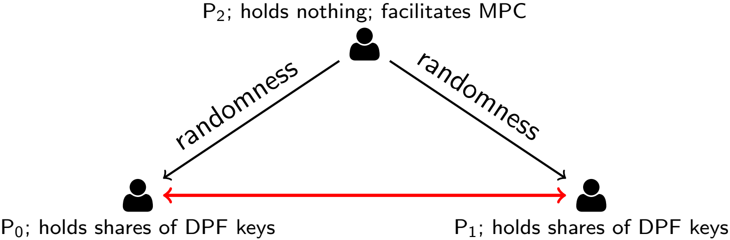
1. **Gets:** $\mathcal{H}(M_i[0,1])$, $\mathcal{H}(M_i[1,0])$, seed_i
2. **Computes:** $M_i[2,0]$; i.e. ordered pair of messages from $\mathsf{P}_2 \rightarrow \mathsf{P}_0$ and $M_i[2,1]$.



Verifier 0, i^{th} iteration (the other verifier is symmetrical)

Case B, $c_i = 1$; Given that P_2 follows the protocol do P_0 and P_1 follow the protcol?

1. **Gets:** $\mathcal{H}(M_i[2,1])$, $M_i[0,1]$ and $M_i[2,0]$; i.e. ordered pair of messages from $\mathsf{P}_0 \rightarrow \mathsf{P}_1$ and $\mathsf{P}_2 \rightarrow \mathsf{P}_0$.
2. **Computes:** $M_i[1,0]$; i.e. ordered pair of messages from $\mathsf{P}_1 \rightarrow \mathsf{P}_0$.



Reconstructing the Merkle-tree

Verifier 0

- Verifier 0 has $\mathcal{H}(M_i[0,1])$, $\mathcal{H}(M_i[1,0])$, $\mathcal{H}(M_i[2,1])$, $\mathcal{H}(M_i[2,0])$ for all i.
- Thus, it can compute the root of the merkle-tree.

Intuition behind why this works

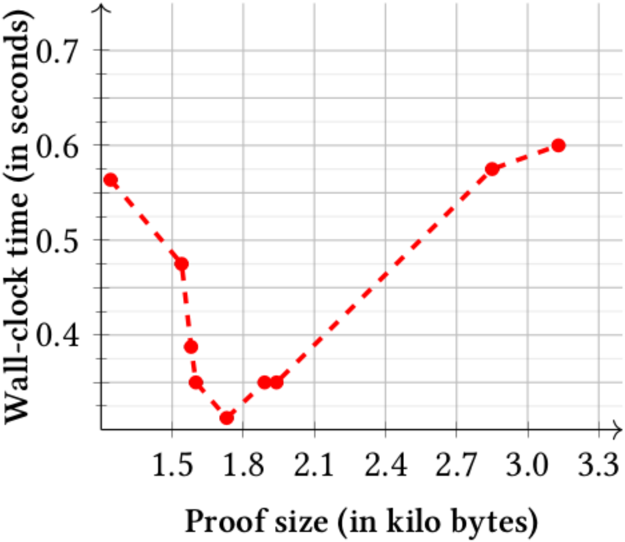
- For each iteration we either checking if P_2 follows the protocol or
- Given that P_2 follows the protocol, do P_0 and P_1 follow the protocol.

Since, the prover has no way to know what would be checked in a particular iteration, the probability or cheating becomes low.

Experiments; 2 Verifier Sabre

size	prooftime
2^{30}	0.64
2^{28}	0.57
2^{26}	0.43
2^{24}	0.37
2^{22}	0.22

Experiments; 2 Verifier Sabre



4-Party Sanity Check

2 Verifier Sabre

has to use LowMC block cipher in order to do the MPC.

AES Block Cipher

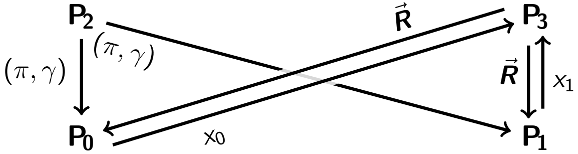
We present our 4-Party sanity check which can use the AES block cipher.

Main Idea

1. We want to verify that the evaluation vector of the two DPFs differ at exactly one location (i.e. they are shares of a standard basis vector).
2. P_3 sends a random vector $\vec{\boldsymbol{R}}$ to P_0 and P_1.
3. P_b compute $\mathsf{out}_b \leftarrow \oplus_{\mathsf{Evalfull}(\mathsf{dpf}_b)[i]=1} \vec{\boldsymbol{R}}[i]$ and send to P_2.
4. P_2 verifies that $\mathsf{out}_0 \oplus \mathsf{out}_1 \in \vec{\boldsymbol{R}}$

4-Party Sanity Check

Output: $\begin{cases} \texttt{accept if } \exists j,\ x_0 \oplus x_1 \overset{?}{=} \vec{\boldsymbol{R}}_j; \\ \texttt{reject otherwise.} \end{cases}$



$x_0 \leftarrow \left(\bigoplus_{\mathsf{flags}(\mathsf{seed}_0)[i]=1} \vec{\boldsymbol{R}}[\pi(i)]\right) \oplus \gamma \qquad x_1 \leftarrow \left(\bigoplus_{\mathsf{flags}(\mathsf{seed}_1)[i]=1} \vec{\boldsymbol{R}}[\pi(i)]\right)$

Downsides

1. Probabilistic.
2. Requires 4 Parties.

Experiments; 4P Sanity Check