Amortizing Secure Computation with Penalties

Iddo Bentov
Cornell University

Ranjit Kumaresan
MIT

CCS 2016

Message

Takeaway message

- A new variant of off-chain channels:
- Off-chain channels are useful not only for (micro) payments.
  - Instantaneous fair exchange (of verifiable data), with penalties
  - Instantaneous fair secure computation, with penalties.

Message

Takeaway message

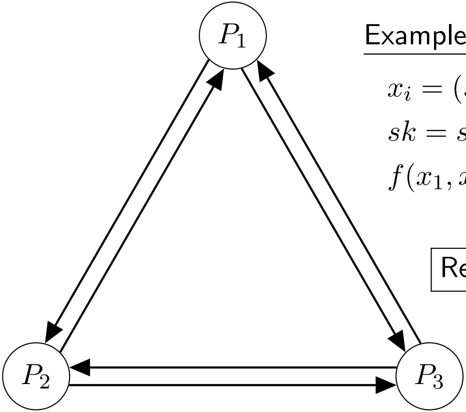
- A new variant of off-chain channels:
- Off-chain channels are useful not only for (micro) payments.
  - Instantaneous fair exchange (of verifiable data), with penalties
  - Instantaneous fair secure computation, with penalties.

How expressive should the scripting language be?

- New use-case for an opcode that verifies arbitrary signatures.
- Different use-cases for this opcode:
  - lottery-based micropayments [Pass, shelat: CCS15]
  - anonymous transactions [Heilman, Baldimtsi, Goldberg: FC16]

Secure multiparty computation (MPC) / secure function evaluation (SFE)

Parties $P_1, P_2, \ldots, P_n$ with inputs $x_1, x_2, \ldots, x_n$ send messages to each other, and wish to ***securely*** compute $f(x_1, x_2, \ldots, x_n)$.



Example of SFE:

$$x_i = (sk_i, c)$$

$$sk = sk_1 \oplus sk_2 \oplus \cdots \oplus sk_n$$

$$f(x_1, x_2, \ldots, x_n) = \texttt{decrypt}(sk, c)$$

Reactive MPC: think of poker cards

Impossibility of fair MPC in the standard communication model

Fairness: if any party receives the output, then all honest parties must receive the output.

"Security with abort" is possible

- Secure MPC is possible [Yao86, GMW87, ...]
  - Security: correctness, privacy, independence of inputs, ~~fairness~~
  - Even with dishonest majority, in the computational setting.

Full security is impossible

- Fair MPC is impossible [Cle86]
  - r-round 2-party coin toss protocol is susceptible to $\Omega(1/r)$ bias.
  - $\Rightarrow$ no fair protocol for XOR, barring gradual release [...]

Overview

This presentation

1. Impose fairness for any SFE, without an honest majority.
2. For 2 parties, ℓ sequential executions of (different) fair SFE with only two $\mathcal{F}^{\star}_{\mathrm{CR}}$ invocations, instead of $\Omega(\ell)$ invocations.
3. For n parties and r-rounds reactive MPC, $O(n^2 r)$ invocations.

Not in this presentation

- Secure cash distribution (e.g., poker).

Formal model that incorporates coins

Functionality $\mathcal{F}_\square$ versus functionality $\mathcal{F}^\star_\square$ with coins

- If party P_i has some secret s_0 and sends it to party P_j, then both P_i and P_j will have the string s_0.
- If party P_i has coins(x) and sends $y < x$ coins to party P_j, then P_i will have coins$(x - y)$ and P_j will have extra coins(y).

- With Bitcoin: the parties only send strings, but miners do PoW so that the coin transfers become irreversible.

Formal model that incorporates coins

Functionality $\mathcal{F}_\square$ versus functionality $\mathcal{F}^\star_\square$ with coins

- If party P_i has some secret s_0 and sends it to party P_j, then both P_i and P_j will have the string s_0.
- If party P_i has coins(x) and sends $y < x$ coins to party P_j, then P_i will have coins$(x - y)$ and P_j will have extra coins(y).

- With Bitcoin: the parties only send strings, but miners do PoW so that the coin transfers become irreversible.
- Ideally, all the parties deem coins to be valuable assets.
- Sending coins(x) may require a broadcast that reveals at least the amount x and pseudonyms (not in ZK/anon cryptocurrency).
- We provide simulation based proofs (not in this talk).

Claim-or-Refund for two parties P_s,P_r (implicit in [Max11],[BBSU12])

The $\mathcal{F}^{\star}_{\mathrm{CR}}$ Claim-or-Refund ideal functionality

1. The sender P_s deposits (locks) her coins(q) while specifying a time bound τ and a circuit $\phi(\cdot)$.
2. The receiver P_r can claim (gain possession) of the coins(q) by publicly revealing a witness w that satisfies $\phi(w) = 1$.
3. If P_r didn't claim within time τ, coins(q) are refunded to P_s.

How to realize $\mathcal{F}^{\star}_{\mathrm{CR}}$ via Bitcoin

- Old version: using "timelock" transactions.
- New version: `OP_CHECKLOCKTIMEVERIFY` (abbrv. `CLTV`) enables $\mathcal{F}^{\star}_{\mathrm{CR}}$ directly, avoiding transaction malleability attacks.

$\mathcal{F}^{\star}_{\mathrm{CR}}$ via Bitcoin with CLTV (operational since ≈ December 2015)

Pseudocode: pk_S, pk_R, h_0, τ are hardcoded

if (block# > τ) then
    P_s can spend the coins(q) by signing with sk_s
else
    P_r can spend the coins(q) by
        signing with sk_r
        AND
        supplying w such that Hash(w) $= h_0$ $\boxed{\leftarrow \text{this is } \phi(\cdot)}$

Bitcoin script

```
IF <timeout> CHECKLOCKTIMEVERIFY OP_DROP <pk_s>
CHECKSIGVERIFY ELSE HASH256 <h_0> EQUALVERIFY <pk_r>
CHECKSIGVERIFY ENDIF
```

Fairness with penalties (non-reactive)

Definition of fair secure multiparty computation with penalties

- An honest party never has to pay any penalty
- If a party aborts after learning the output and doesn't deliver output to honest parties $\Rightarrow$ every honest party is compensated

Fairness with penalties (non-reactive)

Definition of fair secure multiparty computation with penalties

- An honest party never has to pay any penalty
- If a party aborts after learning the output and doesn't deliver output to honest parties $\Rightarrow$ every honest party is compensated

Outline of $\mathcal{F}_f^\star$ – fairness with penalties for any function f

- $P_1, \ldots, P_n$ with $x_1, \ldots, x_n$ run secure *unfair* SFE for f that
  1. Computes random $y_1 \oplus y_2 \oplus \cdots \oplus y_n = y$ for $y = f(x_1, \ldots, x_n)$
  2. Computes $\mathrm{Tags} = (\mathsf{com}(y_1), \ldots, \mathsf{com}(y_n))$ $\boxed{= (\mathtt{hash}(y_1), \ldots, \mathtt{hash}(y_n))}$
  3. Delivers (y_i, Tags) to every P_i
- $P_1, \ldots, P_n$ deposit coins and run fair exchange with penalties to swap the y_i's among themselves.

Fair exchange in the $\mathcal{F}^{\star}_{\mathrm{CR}}$-hybrid model - the ladder construction

"Abort" attack:

P_2 claims without deposting

$$P_1 \xrightarrow[q,\tau]{w_2} P_2$$
$$P_2 \xrightarrow[q,\tau]{w_1} P_1$$

Fair exchange:

P_1 claims by revealing w_1

$\Rightarrow$ P_2 can claim by revealing w_2

$$\boxed{\tau_1 < \tau_2}$$
$$P_1 \xrightarrow[q,\tau_2]{w_1 \wedge w_2} P_2$$
$$P_2 \xrightarrow[q,\tau_1]{w_1} P_1$$

Malicious coalition:

Coalition P_1, P_2 obtain w_3 from P_3

P_2 doesn't claim the top transaction

P_3 isn't compensated

$$\boxed{\tau_1 < \tau_2 < \tau_3}$$
$$P_1 \xrightarrow[q,\tau_3]{w_1 \wedge w_2 \wedge w_3} P_2$$
$$P_2 \xrightarrow[q,\tau_2]{w_1 \wedge w_3} P_3$$
$$P_3 \xrightarrow[q,\tau_1]{w_1} P_1$$

Fair exchange in the $\mathcal{F}^{\star}_{\text{CR}}$-hybrid model - the ladder construction (contd.)

Fair exchange:

Bottom two levels:

P_1, P_2 get compensated by P_3

Top two levels:

P_3 gets her refunds by revealing w_3

$$\boxed{\tau_1 < \tau_2 < \tau_3}$$

$$P_1 \xrightarrow[q,\tau_3]{w_1 \wedge w_2 \wedge w_3} P_3$$

$$P_2 \xrightarrow[q,\tau_3]{w_1 \wedge w_2 \wedge w_3} P_3$$

$$P_3 \xrightarrow[2q,\tau_2]{w_1 \wedge w_2} P_2$$

$$P_2 \xrightarrow[q,\tau_1]{w_1} P_1$$

Full ladder:

Roof Deposits.

$$P_1 \xrightarrow[q,\tau_n]{T_1 \wedge \cdots \wedge T_n} P_n$$

$$P_2 \xrightarrow[q,\tau_n]{T_1 \wedge \cdots \wedge T_n} P_n$$

$$\vdots$$

$$P_{n-2} \xrightarrow[q,\tau_n]{T_1 \wedge \cdots \wedge T_n} P_n$$

$$P_{n-1} \xrightarrow[q,\tau_n]{T_1 \wedge \cdots \wedge T_n} P_n$$

Ladder Deposits.

$$P_n \xrightarrow[(n-1)q,\tau_{n-1}]{T_1 \wedge \cdots \wedge T_{n-1}} P_{n-1}$$

$$P_{n-1} \xrightarrow[(n-2)q,\tau_{n-2}]{T_1 \wedge \cdots \wedge T_{n-2}} P_{n-2}$$

$$\vdots$$

$$P_3 \xrightarrow[2q,\tau_2]{T_1 \wedge T_2} P_2$$

$$P_2 \xrightarrow[q,\tau_1]{T_1} P_1$$

Comparison with other ways to achieve fairness

Gradual release

- Release the output bit by bit...
- Even with only 2 parties, the number of rounds depends on a security parameter.
- Complexity blowup because the protocol must ensure that the parties don't release junk bits.
- Assumptions on the computational power of the parties, sequential puzzles to avoid parallelization.

Fairness with penalties

- With Bitcoin, the PoW miners do all the heavy lifting.
- Still, we don't want to wait for on-chain PoW confirmations...

Amortized protocol – what we achieve

- Unbounded number of sequential MPC executions, with **off-chain** fair exchange (with penalties) of the outputs, as long as all parties are honest.
- Resembles optimistic fair exchange, but with no trusted party.

Main idea

Since the (commitments to the) output values are not known in advance, the $\mathcal{F}^{\star}_{\mathrm{CR}}$ on-chain transactions require the parties to reveal signatures of indexed messages.

The general case: amortized reactive secure-MPC

- Multistage protocol: after each stage of the computation some intermediate outputs are revealed to the parties.
  - Example: the top card of the deck is revealed to all parties.
- One-shot protocol is not the natural formulation:
  - A circuit that takes into account all the possible variables is highly inefficient.
  - Those variables may depend on external events (say, you receive a phone call regarding an unrelated financial loss).
- $\Rightarrow$ must be dropout-tolerant:
  - After a stage that reveals information, corrupt parties must be penalized if they abort.
  - In fact, the corrupt parties must be penalized unless they continue the next stage of the computation.

Ingredient #1: see-saw construction (2-party m-rounds illustration)

Roof deposit.

$$P_1 \xrightarrow[q,\tau_{m,2}]{\mathrm{TT}_{m,2}} P_2 \qquad (\mathsf{Tx}_{m,2})$$

See-saw deposits. For $r = m-1$ to 1:

$$P_2 \xrightarrow[2q,\tau_{r+1,1}]{\mathrm{TT}_{r+1,1}} P_1 \qquad (\mathsf{Tx}_{r+1,1})$$

$$P_1 \xrightarrow[2q,\tau_{r,2}]{\mathrm{TT}_{r,2}} P_2 \qquad (\mathsf{Tx}_{r,2})$$

Floor deposit.

$$P_2 \xrightarrow[q,\tau_{1,1}]{\mathrm{TT}_{1,1}} P_1 \qquad (\mathsf{Tx}_{1,1})$$

Ingredient #2: circuits that verify signed data

- On-chain $\mathcal{F}^{\star}_{\mathrm{CR}}$ circuits that verify a signed transcript of an execution.
- For a feasibility result, consider signatures that are created inside the secure computation.

$$\phi^{\mathsf{lock}}_{j,i}(\mathrm{TT}, \mathsf{id}, \sigma; mvk) = \mathsf{tv}^{(\mathsf{id})}_{i-1}(\mathrm{TT}) \bigwedge \mathsf{SigVerify}(mvk, (j, i, \mathsf{id}), \sigma)$$

$$\phi_i(\mathrm{TT}, \mathsf{id}; mvk) = \mathsf{tv}^{(\mathsf{id})}_{i}(\mathrm{TT})$$

$$\phi^{\mathsf{unlock}}_{j,i}(\mathrm{TT}, \mathsf{id}, \sigma; mvk) = \mathsf{tv}^{(\mathsf{id})}_{i}(\mathrm{TT}) \bigwedge \mathsf{SigVerify}(mvk, (j, i, \mathsf{id}), \sigma)$$

where $\mathrm{TT} = (T_1^{(\mathsf{id}_1)}, \sigma_1^{(\mathsf{id}_1)}) \| \cdots \| (T_i^{(\mathsf{id}_i)}, \sigma_i^{(\mathsf{id}_i)})$ and $\mathsf{tv}^{(\mathsf{id})}_i(\mathrm{TT}) = 1$ iff

- $\mathsf{id}_1 = \cdots = \mathsf{id}_i \geq \mathsf{id}$.
- for all $j \leq i$: $T_j^{(\mathsf{id}_j)}$ is a message of the form $(j, \mathsf{id}_j, *)$ and $\sigma_j^{(\mathsf{id}_j)}$ is a valid signature on $T_j^{(\mathsf{id}_j)}$ under msk.

Ingredient #3: multiparty "locked" ladder

LADDER DEPOSITS. For $i = n-1$ down to 1:

- Rung unlock: For $j = n$ down to $i+1$:

$$P_j \xrightarrow[q,\tau_{j,i}^{\text{unlock}}]{\phi_{j,i}^{\text{unlock}}} P_i$$

- Rung climb:

$$P_{i+1} \xrightarrow[i \cdot q,\tau_i]{\phi_i} P_i$$

- Rung lock: For each $j = n$ down to $i+1$:

$$P_i \xrightarrow[q,\tau_{j,i}^{\text{lock}}]{\phi_{j,i}^{\text{lock}}} P_j$$

Amortized reactive secure MPC - summary

Work	Case	$\mathcal{F}^{\star}_{\text{CR}}$ calls	Max deposit	Script comp.†	Round comp.*	Assump.
Crypto14	One-shot	$O(n\ell)$	$O(nq)$	$O(n^2 z\ell)$	$O(n\ell)$	owf,$\mathcal{F}_{\text{OT}}$
CCS16	One-shot	$O(n\ell)$	$O(nq)$	$O(n\lambda\ell)$	$O(n\ell)$	RO,$\mathcal{F}_{\text{OT}}$
Ours	One-shot	$O(n^2)$	$O(nq)$	$O(n^3 z)$	$O(n)$	owf,$\mathcal{F}_{\text{OT}}$
CCS15	Reactive	$O(n^2 r)$	$O(nq)$	$O(n^2 T\ell)$	$O(nr)$	etdp
CCS16	Reactive	$O(nr\ell)$	$O(nr^2 q)$	$O(nT\ell)$	$O(nr\ell)$	etdp
Ours	Reactive	$O(n^2 r)$	$O(nrq)$	$O(n^2 T)$	$O(nr)$	etdp

Table: n: number of parties; q: penalty amount; z: length of output of f (we assume $z \gg \lambda$); λ: computational security parameter; T (resp. r): size of transcript (resp. number of rounds) of an n-party secure computation protocol that implements f in the plain model; owf: one-way functions; $\mathcal{F}_{\text{OT}}$: ideal oblivious transfer; RO: random oracle; etdp: enhanced trapdoor permutations; Note that ℓ is a parameter, thus our costs *per execution* tend to zero as ℓ grows. The '*' in the round complexity column means that the values in the column refer to the "on-chain round complexity." The "off-chain round complexity" of our protocol is $O(n\ell)$ in the one-shot case and $O(nr\ell)$ in the reactive case.

Amortized protocol for 2 parties

Note: this is a portion from a followup work.

Preparation:

1. P_1 make an $\mathcal{F}^{\star}_{\mathrm{CR}}$ transaction to P_2 with q coins, timeout τ_1, and circuit $\phi_1(m_1, m_2, H_1, H_2, S_1, S_2)$ that
   - Parses $H_1 = (i, h_1),\ H_2 = (j, h_2)$
   - Verifies $i = j,\ m_1 \neq m_2,\ \mathsf{Hash}(m_1) = h_1,\ \mathsf{Hash}(m_2) = h_2$
   - Verifies signatures: $\mathsf{SigVerify}_{pk_1}(H_1, S_1),\ \mathsf{SigVerify}_{pk_1}(H_2, S_2)$
2. P_2 make an $\mathcal{F}^{\star}_{\mathrm{CR}}$ transaction to P_1 with q coins, timeout $\tau_2 < \tau_1$, and circuit $\phi_2(m, H, S_1, S_2)$ that
   - Parses $H = (\llcorner\!\lrcorner, h)$ and verifies that $\mathsf{Hash}(m) = h$
   - Verifies signatures: $\mathsf{SigVerify}_{pk_1}(H, S_1),\ \mathsf{SigVerify}_{pk_2}(H, S_2)$

Amortized protocol for 2 parties (contd.)

Executions:

3. Until time τ_2, P_1 and P_2 execute any number of SFE invocations with functions $f_i(x_1, x_2), i = 1, 2, \ldots,$ such that
   - $y_i = f_i(x_{i,1}, x_{i,2})$, and $m_{i,1} \oplus m_{i,2} = y_i$ are additive shares of y_i.
   - Commitments: $h_{i,1} = \text{Hash}(m_{i,1}),\ h_{i,2} = \text{Hash}(m_{i,2})$
   - P_1's output is $(m_{i,1}, h_{i,1}, h_{i,2})$, P_2's output is $(m_{i,2}, h_{i,1}, h_{i,2})$.
4. Then, for each execution i,
   - Denote $H_{i,1} = (i, h_{i,1}),\ H_{i,2} = (i, h_{i,2})$.
   - P_1 sends $S_{i,1,2} = \text{Sign}_{sk_1}(H_{i,2})$ to P_2.
   - P_2 runs $\text{SigVerify}_{pk_1}(H_{i,2}, S_{i,1,2})$, and sends $S_{i,2,1} = \text{Sign}_{sk_2}(H_{i,1})$ to P_1.
   - P_1 sends $m_{i,1}$ to P_2, and waits for a short timeout to receive $m_{i,2}$ from P_2.
   - If $m_{i,2}$ was not received, P_1 redeems q coins by revealing $S_{i,1,1} = \text{Sign}_{sk_1}(H_{i,1})$ to satisfy ϕ_2.
   - P_2 can now use $(S_{i,1,1}, S_{i,1,2})$ with $m_{i,2}$ to redeem q coins too.

Amortized protocol for 2 parties - order of events

P_1 needs $\boxed{m, S_1(\mathsf{Hash}(m)), S_2(\mathsf{Hash}(m))}$ to collect the money.

P_2 needs $\boxed{m_1, m_2, S_1(i, \mathsf{Hash}(m_1)), S_1(i, \mathsf{Hash}(m_2))}$ to collect.

$$
\begin{tikzcd}
\textbf{start:}\ \boxed{P_1} \arrow[rr, "{S_1(i, \mathsf{Hash}(m_2))}"] & & \boxed{P_2} \arrow[d, "{S_2(i, \mathsf{Hash}(m_1))}"] \\
\boxed{P_2} \arrow[u, "m_2"] & & \boxed{P_1} \arrow[ll, "m_1"]
\end{tikzcd}
$$

What if P_2 aborts instead of sending m_2 ?

P_1 reveals $\boxed{m_1, S_1(\mathsf{Hash}(m_1))}$ with $S_2(i, \mathsf{Hash}(m_1))$ to collect.

P_2 reveals $\boxed{m_2}$ with $m_1, S_1(\mathsf{Hash}(m_1)), S_1(i, \mathsf{Hash}(m_2))$ to recoup.

Amortized protocol for 2 parties - properties

- P_1 reveals a signed message with a corresponding preimage in every execution i, but P_2 cannot recycle an old signed message to avoid revealing the current output, because the indices won't match.
- P_2 needs to keep a backlog of the signed messages from all the previous executions, but has the advantage of being able to pay q coins to learn the output ($q' = q + \varepsilon$ in ϕ_1 is also possible).
- The scripts ϕ_1, ϕ_2 need an opcode for arbitrary signature verification - same complexity as the standard `CHECKSIGVERIFY`.

Thank you.