Amortizing Secure Computation with Penalties

Iddo Bentov  
Cornell University

Ranjit Kumaresan  
MIT

CCS 2016
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.
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, Baldiviatsi, 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:

\[
x_i = (sk_i, c) \\
sk = sk_1 \oplus sk_2 \oplus \cdots \oplus sk_n \\
f(x_1, x_2, \ldots, x_n) = \text{decrypt}(sk, c)
\]

Reactive MPC: think of poker cards
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 [...]

Impossibility of fair MPC in the standard communication model
Overview

This presentation

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

Not in this presentation

- Secure cash distribution (e.g., poker).
Formal model that incorporates coins

**Functionality $F$ versus functionality $F^*$ 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 $F_2$ versus functionality $F^*$ 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 $\mathcal{F}_{CR}^*$ Claim-or-Refund ideal functionality

1. The sender $P_s$ deposits (locks) her coins$(q)$ while specifying a time bound $\tau$ 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 $\tau$, coins$(q)$ are refunded to $P_s$.

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

- Old version: using “timelock” transactions.
- New version: OP_CHECKLOCKTIMEVERIFY (abbrv. CLTV) enables $\mathcal{F}_{CR}^*$ directly, avoiding transaction malleability attacks.
**F*\textsubscript{CR} via Bitcoin with CLTV (operational since \(\approx\) December 2015)**

**Pseudocode:** \(pk_S, pk_R, h_0, \tau\) are hardcoded

if \((\text{block}\# > \tau)\) 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 \(\text{Hash}(w) = h_0\) \[\text{this is } \phi(.)\]

**Bitcoin script**

```
IF <timeout> CHECKLOCKTIMEVERIFY OP_DROP <pk_s>
CHECKSIGVERIFY ELSE HASH256 <h_0> EQUALVERIFY <pk_r>
CHECKSIGVERIFY ENDFI
```
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
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 \( F^*_f \) – 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 \( \text{Tags} = (\text{com}(y_1), \ldots, \text{com}(y_n)) = (\text{hash}(y_1), \ldots, \text{hash}(y_n)) \)
  3. Delivers \((y_i, \text{Tags})\) 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 $\mathcal{F}_{\text{CR}}^*$-hybrid model - the ladder construction

"Abort" attack:
P₂ claims without depositing

Fair exchange:
P₁ claims by revealing $w₁$
⇒ P₂ can claim by revealing $w₂$

Malicious coalition:
Coalition P₁, P₂ obtain $w₃$ from P₃
P₂ doesn't claim the top transaction
P₃ isn't compensated
Fair exchange in the $\mathcal{F}^*_\text{CR}$-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 $w_3$

**Full ladder:**

<table>
<thead>
<tr>
<th>Roof Deposits.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1 \xrightarrow{T_1 \land \cdots \land T_n \ q, \tau_n} P_n$</td>
</tr>
<tr>
<td>$P_2 \xrightarrow{T_1 \land \cdots \land T_n \ q, \tau_n} P_n$</td>
</tr>
<tr>
<td>$\vdots$</td>
</tr>
<tr>
<td>$P_{n-2} \xrightarrow{T_1 \land \cdots \land T_n \ q, \tau_n} P_n$</td>
</tr>
<tr>
<td>$P_{n-1} \xrightarrow{T_1 \land \cdots \land T_n \ q, \tau_n} P_n$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ladder Deposits.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_n \xrightarrow{T_1 \land \cdots \land T_{n-1} \ (n-1)q, \tau_{n-1}} P_{n-1}$</td>
</tr>
<tr>
<td>$P_{n-1} \xrightarrow{T_1 \land \cdots \land T_{n-2} \ (n-2)q, \tau_{n-2}} P_{n-2}$</td>
</tr>
<tr>
<td>$\vdots$</td>
</tr>
<tr>
<td>$P_3 \xrightarrow{T_1 \land T_2 \ 2q, \tau_2} P_2$</td>
</tr>
<tr>
<td>$P_2 \xrightarrow{T_1 \ q, \tau_1} P_1$</td>
</tr>
</tbody>
</table>

Figure 6: Roof and Ladder deposit phases for fair reconstruction.
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 $\mathcal{F}^*_\text{CR}$ 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).
- ⇒ 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.**

$$
P_1 \xrightarrow{TT_{m,2}} P_2 \\
q,\tau_{m,2} \quad (T_{x_{m,2}})
$$

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

$$
P_2 \xrightarrow{TT_{r+1,1}} P_1 \\
2q,\tau_{r+1,1} \quad (T_{x_{r+1,1}})
$$

$$
P_1 \xrightarrow{TT_{r,2}} P_2 \\
2q,\tau_{r,2} \quad (T_{x_{r,2}})
$$

**Floor deposit.**

$$
P_2 \xrightarrow{TT_{1,1}} P_1 \\
q,\tau_{1,1} \quad (T_{x_{1,1}})$$
Ingredient #2: circuits that verify signed data

- On-chain $\mathcal{F}^*_\text{CR}$ circuits that verify a signed transcript of an execution.

- For a feasibility result, consider signatures that are created inside the secure computation.

\[
\begin{align*}
\phi_{j,i}^{\text{lock}}(TT, id, \sigma; mvk) &= tv_{i-1}^{(id)}(TT) \land \text{SigVerify}(mvk, (j, i, id), \sigma) \\
\phi_i^{(id)}(TT, id; mvk) &= tv_i^{(id)}(TT) \\
\phi_{j,i}^{\text{unlock}}(TT, id, \sigma; mvk) &= tv_i^{(id)}(TT) \land \text{SigVerify}(mvk, (j, i, id), \sigma)
\end{align*}
\]

where $TT = (T_1^{(id_1)}, \sigma_1^{(id_1)}) \parallel \cdots \parallel (T_i^{(id_i)}, \sigma_i^{(id_i)})$ and $tv_i^{(id)}(TT) = 1$ iff

- $id_1 = \cdots = id_i \geq id$.

- for all $j \leq i$: $T_j^{(id_j)}$ is a message of the form $(j, id_j, \ast)$ and $\sigma_j^{(id_j)}$ is a valid signature on $T_j^{(id_j)}$ under $msk$. 
**Ladder Deposits.** For $i = n - 1$ down to 1:

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

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

- **Rung climb:**

  \[
P_{i+1} \xrightarrow{\phi_i} P_i
  \]

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

  \[
P_i \xrightarrow{\phi_{j,i}^{\text{lock}}} P_j
  \]
### Amortized reactive secure MPC - summary

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

**Table:** $n$: number of parties; $q$: penalty amount; $z$: length of output of $f$ (we assume $z \gg \lambda$); $\lambda$: 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; $\mathcal{F}_{OT}$: ideal oblivious transfer; $RO$: random oracle; $etdp$: enhanced trapdoor permutations; Note that $\ell$ is a parameter, thus our costs *per execution* tend to zero as $\ell$ 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 $\mathcal{F}_{\text{CR}}^*$ transaction to $P_2$ with $q$ coins, timeout $\tau_1$, 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 $i = j$, $m_1 \neq m_2$, $\text{Hash}(m_1) = h_1$, $\text{Hash}(m_2) = h_2$
   - Verifies signatures: $\text{SigVerify}_{pk_1}(H_1, S_1)$, $\text{SigVerify}_{pk_1}(H_2, S_2)$

2. $P_2$ make an $\mathcal{F}_{\text{CR}}^*$ transaction to $P_1$ with $q$ coins, timeout $\tau_2 < \tau_1$, and circuit $\phi_2(m, H, S_1, S_2)$ that
   - Parses $H = (\text{\_\_}, h)$ and verifies that $\text{Hash}(m) = h$
   - Verifies signatures: $\text{SigVerify}_{pk_1}(H, S_1)$, $\text{SigVerify}_{pk_2}(H, S_2)$
Executions:

3 Until time $\tau_2$, $P_1$ 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 $\phi_2$.
   - $P_2$ 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 $m, S_1(\text{Hash}(m)), S_2(\text{Hash}(m))$ to collect the money.

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

What if $P_2$ aborts instead of sending $m_2$?

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

$P_2$ reveals $m_2$ with $m_1, S_1(\text{Hash}(m_1)), S_1(i, \text{Hash}(m_2))$ 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 $\phi_1$ is also possible).

- The scripts $\phi_1, \phi_2$ need an opcode for arbitrary signature verification - same complexity as the standard CHECKSIGVERIFY.
Thank you.