“Tiny OT” – Part 3

A New (4 years old) Approach to Practical Active-Secure Two-Party Computation

Claudio Orlandi, Aarhus University
TinyOT authenticated bits

- $[x] = ( (x_A, k_A, m_A), (x_B, k_B, m_B) )$ s.t.
  - $m_B = k_A + x_B \Delta_A$ (symmetric for $m_A$)
  - $\Delta_A, \Delta_B$ is the same for all wires.
  - MACs, keys are $k$-bit strings.

(Maybe adversary knows a few bits of $\Delta$)

Similarity with Oblivious Transfer

- Sender has two messages $u_0, u_1$
- Receiver has a bit $b$ and learns $u_b$
- Set $u_0 = k$, $u_1 = k + \Delta$, $b = x$ then $u_b = k + x\Delta$
Recap

1. **Output Gates:**
   - Exchange shares and MACs
   - Abort if MAC does not verify

2. **Input Gates:**
   - Get a random \([r]\) from *trusted dealer*
   - \(r \leftarrow \text{Open}(A,[r])\)
   - Alice sends Bob \(d=x-r\),
   - Compute \([x]=[r]+d\)
Recap

1. Addition Gates:
   - Use linearity of representation to compute
     \[ z = [x] + [y] \]

2. Multiplication gates:
   - Get a random triple \([a][b][c]\) with \(c = ab\) from TD.
   - \(e \leftarrow \text{Open}([a]+[x]), \ d \leftarrow \text{Open}([b]+[y])\)
   - Compute \([z] = [c] + a[y] + b[x] - ed\)
Circuit Evaluation (Online phase)

3) $[z] \leftarrow \text{Mul}([x],[y])$:

- Get $[a],[b],[c]$ with $c=ab$ from trusted dealer

- $e=\text{Open}([a]+[x])$
- $d=\text{Open}([b]+[y])$

- Compute $[z] = [c] + e[y] + d[x] - ed$
  
  $ab + (ay+xy) + (bx+xy) - (ab+ay+bx+xy)$
Coming up...

- Given **authenticated bits**, produce
  **authenticated multiplication triples**!
The problem

- **Input:** (random) \([x], [y], [r], [s], ...\)
- **Output:** \([z]\) s.t. \([z=xy]\)

\[
= x_A y_A + x_A y_B + x_B y_A + x_B y_B
\]

**Remember**

- \([x] = (x_A, k_A, m_A), (x_B, k_B, m_B)\) s.t.
- \(m_B = k_A + x_B \Delta_A\) (symmetric for \(m_A\))
- \(\Delta_A, \Delta_B\) is the same for all wires.
- MACs, keys are \(k\)-bit strings.

How to authenticate local product?

How to authenticate cross product?
Part 3:
From “Auth. Bits” to “Auth. Triples”

• Authenticated local-products ($a\text{AND}$)

• Authenticated cross-products ($aOT$)

• “LEGO” bucketing
Authenticate local products

• **Input:** \([x], [y], [r]; \) **Alice private input:** \(x, y\)

• **Output:** \([z]\) s.t. \(z = xy\)

• **First Attempt:** (like Input)
  
  – \(r \leftarrow \text{Open}(A,[r])\)
  
  – Alice sends Bob \(d = r + xy + e\)
  
  – \([z]=[xy]+r + e\)

• **Corrupted Alice, what if \(e \neq 0\) ?**
Authenticate local products

- $\Delta$ is the same for all wires.
- $[x] = ( (x,...,m_x), (...,k_x,...) )$ s.t. $m_x = k_x + x \Delta$
- $[y] = ( (y,...,m_y), (...,k_y,...) )$ s.t. $m_y = k_y + y \Delta$
- $[z] = ( (z,...,m_z), (...,k_z,...) )$ s.t. $m_z = k_z + z \Delta$

- When $x = 0$
  
  $\left( m_x = k_x, m_z = k_z \right)$ iff $z = 0$

- When $x = 1$
  
  $\left( m_x = k_x + \Delta, m_z + m_y = k_z + k_y \right)$ iff $z = y$
Authenticate local products

- **Bob knows**
  \[ U_0 = (k_x, k_z) \text{ and } U_1 = (k_x + \Delta, k_z + k_y) \]
- **Alice knows**
  \[ U_x \quad \text{if } xy = z \]
  \[ \text{neither} \quad \text{if } xy \neq z \]
- **How can Alice prove she knows** \( U_x \) **without revealing** \( x \)?
Proof of 1-out-of-2 strings

\[ U_x \]

\[ B = H(U_0) + H(U_1) \]

if \( x=0 \) \[ A = H(U_x) \]
else \[ A = C + H(U_x) \]

\[ U_0, U_1 \]
Proof of 1-out-of-2 strings

\[ B = H(U_0) + H(U_1) + e \]

if \( x = 0 \) then \( A = H(U_x) \)
else \( A = C + H(U_x) \)

\[ A = H(U_0) + xe \]
Proof of 1-out-of-2 strings

$U_x$

$B = H(U_0) + H(U_1) + e$

$U_0, U_1$

$\text{if}(x=0) \ A = H(U_x)$

$\text{else} \quad A = C + H(U_x)$

\[ B = H(U_0) + H(U_1) + e \]

If $e \neq 0$

w.p. $\frac{1}{2}$ abort with probability $\frac{1}{2}$

w.p. $\frac{1}{2}$ continue and Bob learns $x$

$A$ \hspace{3cm} EQ \hspace{3cm} H(U_0) + xe$

ok/abort \hspace{3cm} ok/abort
Combine local multiplications

- **Input:** (random) \([x_1], [y_1], [z_1], [x_2], [y_2], [z_2]\)
  
  // \(z_i = x_i y_i\), Alice knows all
  
  // Bob knows: \(x_1\) or \(x_2\) (not both)

- **Output:** \([a], [b], [c]\)  // Bob knows nothing

1. \([a] = [x_1] + [x_2]\)  // Now a random

2. \([b] = [y_1]\)

3. \(d = \text{Open}([y_1] + [y_2])\)

4. \([c] = [z_1] + [z_2] + d[x_2]\)

  // \(x_1 y_1 + x_2 y_2 + x_2 y_1 + x_2 y_2 = (x_1 + x_2) y_1 = ab\)

- Authenticated local-products ($aAND$)
- Authenticated cross-products ($aOT$)
- “LEGO” bucketing
The problem

• **Input:** (random) \([x], [y], [r], [s], \ldots\)
• **Output:** \([z]\) s.t. \([z=xy]\)

\[= x_A y_A + x_A y_B + x_B y_A + x_B y_B\]

• **Remember**
  
  − \([x] = ( (x_A, k_A, m_A), (x_B, k_B, m_B) )\) s.t.
  
  − \(m_B = k_A + x_B \Delta_A\) (symmetric for \(m_A\))
  
  − \(\Delta_A, \Delta_B\) is the same for all wires.
  
  − MACs, keys are \(k\)-bit strings.
Use auth. bit to do OT

- Alice knows $x$
- $[x] = (x, ..., m_x), (..., k_x, ...) \text{ s.t. } m_x = k_x + x \Delta$

\[

c_0 = H(k_x) + u_0 \\
c_1 = H(k_x + \Delta) + u_1
\]

\[
u_x = c_x + H(m_x)
\]

$[x]$-OT

\[
\begin{cases}
  u_x & \rightarrow [x]\text{-OT} \\
  u_0, u_1 & \rightarrow u_x
\end{cases}
\]
Authenticated cross-products

• **Input:** \([x], [y], [z], [r]\);
• **Alice has private input:** \(x, r\)
• **Bob has private input:** \(y, z\)
• **Output:** \([s]\)  s.t.  \(s = xy + z\)
Authenticated cross-products

\[ s = xy + z \]

\[ d = r + s \]

\[ [s] = [r] + d \]
Authenticated cross-products

\[ s = xy + z \]

What if \( e \neq 0 \)?

\[ d = r + s + e \]

\[ [s] = [r] + d + e \]
If $e \neq 0$
Alice learns only one $U$ value not both!

$x, r$

$s, U_s$

$[x]$-OT

$z, U_z$

$y+z, U_{y+z}$

$d = r + s + e$

$[s] = [r] + d + e$

$U_{1+s+e}$

$[s]$-OT

$U_1, U_0$

$(U_0, U_1)$
\[ x, r \]

\[ s, U_s \]

\[ d = r + s \]

\[ [s] = [r] + d \]

\[ z, U_z \]

\[ y+z, U_{y+z} \]

\[ U_{1+s} \]

\[ U_1, U_0 \]

\[ (U_0, U_1) \]
\[ s + f(x,e), U_s \]

\[ d = r + s + f(x,e) \]

\[ [s+f(x,e)] = [r]+d \]

\[ U_{1+s} \]

\[ U_{1+e}, U_{0+e} \]

\[ z+e, U_z \]

\[ y+z+e, U_{y+z} \]

\[ x, r \]

\[ y, z \]

Lead to wrong result!

Solution: make sure that cheating leads to aborts w.p. \( \frac{1}{2} \)

Bob learns s!
Step 1:
check $U_0, U_1$ w/EQ
(cheating leads to aborts w.p. $\frac{1}{2}$)
\[ d = r + s + f(x,e) \]

\[ [s+f(x,e)] = [r] + d \]
Step 2: Transfer MAC w/ bit (cheating leads to aborts w.p. ½)

Abort if
\[ m_s \neq k_s + s \Delta \]

\[ d = r + t \]

\[ [s] = [r] + d \]
Abort if
\[ m_s \neq k_s + s \Delta \]

\[ d = r + t \]

\[ [s] = [r] + d \]

\[ e \neq 0 \rightarrow \text{abort w.p. } \frac{1}{2} \]

\[ \rightarrow \text{Learn } x \text{ w.p. } \frac{1}{2} \]
Combine local multiplications

- **Input:** \([x_1], [y_1], [z_1], [s_1], [x_2], [y_2], [z_2], [s_2]\)
  // \(s_i = x_i y_i + z_i\), Alice knows \(x_i, s_i\), Bob knows \(y_i, z_i\)
  // Bob knows: \(x_1 \) or \(x_2\) (not both)

- **Output:** \([a], [b], [c], [t]\)  // Bob knows nothing

1. \([a] = [x_1] + [x_2]\)  // Now a random
2. \([b] = [y_1], [c] = [z_1] + [z_2]\)
3. \(d = \text{Open}([y_1] + [y_2])\)
4. \([t] = [z_1] + [z_2] + d[x_2]\)

// \(x_1 y_1 + z_1 + x_2 y_2 + z_2 + x_2 y_1 + x_2 y_2 = (x_1 + x_2) y_1 + z_1 + z_2 = ab + c\)
Part 3:
From “Auth. Bits” to “Auth. Triples”

• Authenticated local-products ($a\text{AND}$)

• Authenticated cross-products ($a\text{OT}$)

• “LEGO” bucketing
Finishing Up

• We can compute **local-products** and **cross-products** where if **one party cheats**
  – w.p. $\frac{1}{2}$ protocol **aborts**
  – w.p. $\frac{1}{2}$ protocol **continues**
    and cheating party **learns 1 bit**

• If protocol continues
  ➔ There are at most $\sigma$ leaked bits (w.p. $2^{-\sigma}$)
  ➔ Let $M$ #multiplication gates
  ➔ Typically $M \gg \sigma$
“LEGO” bucketing

• Bucket size $B$, $M$ buckets
  – overhead, # of multiplications

• Total work $BM$, randomly assign in buckets
  – # of generated triples

• Secure if $\geq 1$ “good” in each bucket
  – using combiners presented before

• Stat. Sec. $2^{-\sigma}$ with bucket size $B = \frac{\sigma}{\log_2 N}$
  – Larger circuits $\rightarrow$ more efficiency!
Tiny OT - Recap

• Preprocessing
  – Generate authenticated bits (OT extension)
  – Exploit duality authenticated bit/OT to perform local multiplications and cross multiplications efficiently (but with some limited leakage)
  – Randomly assign in small buckets (e.g., B=4)
  – Combine to get rid of leakage

• Online phase
  – Use precomputed triples to evaluate any circuit.