“Tiny OT” – Part 2

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

Claudio Orlandi, Aarhus University
\[(r_A, r_B) \leftarrow D\]

Trusted Dealer

\[f(x, y)\]
Online Phase

Preprocessing

\[ f(x, y) \]
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.

• Very similar to 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 \(u_b = k + x\Delta\)
Two problems:

• **Efficiency**: OT requires public key primitives, inherently efficient
The Crypto Toolbox

OTP >> SKE >> PKE >> FHE >> Obfuscation

More efficient

Less efficient
Two problems:

- **Efficiency**: OT requires public key primitives, inherently efficient

- **Security**: If we authenticated more than one bit, how do we make sure Bob uses the same value $\Delta$?

- Two birds with one stone! Next hour: **Active secure OT extension**!
Authenticated Bits

\[ m_x = k_x + x\Delta \quad \text{and} \quad m_y = k_y + y\Delta \]

\[ z = x + y \quad \text{and} \quad m_z = m_x + m_y \]

\[ [z] = [x] + [y] \]

\[ k_z = k_x + k_y \]

\[ m_z = k_z + z\Delta \]
Authenticated Bits

x

\[ m_x = k_x + x\Delta \]

\[ m_y = k_y + y\Delta + \varepsilon y \]

z = x + y

m_z = m_x + m_y

z, m_z

k_z = k_x + k_y

z = \text{Open}(B, [z])

k_x, k_x + \Delta

(k_y, k_y + \Delta + \varepsilon)

Bob learns y (and therefore x)! (should only learn XOR)
Part 2: Active Secure OT Extension

- Warmup: OT properties
- Recap: Passive Secure OT Extension
- Active Secure OT Extension
• $x_b = x_0 + b(x_0 + x_1)$
• $x_b = (1+b) x_0 + b x_1$
OT = AND

Receiver

b

ab + c

1-2 OT

(a, a+c)

Sender

Bits
Stretching OT

Receiver

\( b \)

\( k_b \)

\( m_b = \text{prg}(k_b) + u_b \)

Sender

\( k_0, k_1 \)

\( m_0, m_1 \)

poly(k)-bit strings

k-bit strings

1-2 OT

\((u_0, u_1) = (\text{prg}(k_0) + m_0), \text{prg}(k_1) + m_1)\)
Random OT = OT

\[(x_0, x_1) = ((r_0 + m_0), (r_1 + m_1))\]

\[m_b = r_c + x_b\]

if \(b = c\)
Random OT = OT

\[(x_0, x_1) = (r_0 + d + m_0, r_1 + d + m_1)\]

Exercise: check that it works!
(R)OT is symmetric

\[ \begin{align*}
  c &= s_0 + s_1 \\
  z &= s_0 \\
  r_0 &= y \\
  r_1 &= b + r_0
\end{align*} \]

Exercise: check that it works

No communication!
Part 2: Active Secure OT Extension

• Warmup: OT properties

• Recap: Passive Secure OT Extension

• Active Secure OT Extension
OT Extension

• OT pro(v/b)ably requires public-key primitives
  – OT extension ≈ hybrid encryption
  – Start from k “real” OTs
  – Turn them into \( \text{poly}(k) \) OTs using only few symmetric primitives per OT
OT Extension, Pictorially

Remember: OT stretching

$k$ \rightarrow \begin{align*} x_{0,1} \quad \text{n=poly}(k) \\ X_0 \quad \text{1-2 OTs} \\ x_{1,1} \quad \text{b} \\ X_1 \end{align*}

$k \rightarrow \begin{align*} 0 \quad \text{U} \\ x_{b1,1} \quad \text{b} \\ 1 \end{align*}$
Condition for OT extension

\[ x_0 \oplus x_1 = \Gamma \]

Problem for active security!
OT Extension, Pictorially

\[(b \otimes \Gamma)_{ij} = b_i \cdot \Gamma_j\]
OT Extension, Turn your head!

\[ U = X_0 \oplus \Gamma \]

\[ b \times \Gamma \]

\[ \leq \]

\[ y \oplus c \]
OT Extension, Pictorially

\[ n = \text{poly}(k) \]
OT Extension, Pictorially

- \( k \)
- \( b \)
- \( U \)
- \( n = \text{poly}(k) \)
- \( X_0 \)
- \( \Gamma \)
- 1-2 OTs

- \( n = \text{poly}(k) \)
Defining $Y_1$

$$Y_1 = Y_0 \oplus \begin{array}{c} \triangle \\ \triangle \\ \triangle \\ \triangle \\ \triangle \end{array}$$
OT Extension, Pictorially

\[ n = \text{poly}(k) \]

\[ Y^1, Y^0 \]

\[ 1-2 \text{ OTs} \]
Finishing Up

• **Problem:** \((Y_0, Y_1)\) not random!

• **Solution:** just hash each row
  - \(Y'_0 = H(Y_0)\)
  - \(Y'_1 = H(Y_1)\)

• Using a *correlation robust hash function* \(H\) s.t.
  1. \(\{a_0, \ldots, a_n, H(a_0 + \Delta), \ldots, H(a_n + \Delta)\}\)
  2. \(\{a_0, \ldots, a_n, b_0, \ldots, b_n\}\) // (a_i’s,b_i’s random)

are *computationally indistinguishable*
OT Extension, Pictorially

\[ n = \text{poly}(k) \]

\[ H(Y) \]

\[ \kappa' \]

\[ \kappa \]

\[ n \]

1-2 OTs
Recap

0. **Stretch** $k$ **OTs** from $k$- to $\text{poly}(k)=n$-bit long strings

1. Set each pair of messages $x^i_0, x^i_1$ s.t. $x^i_0 \oplus x^i_1 = \Gamma$

2. **Turn your head** (S/R swap roles)

3. The bits of $c=\Gamma$ are the new **choice bits**

4. The new messages are of the form $y^{j}_0, y^{j}_1 = y^{j}_0 \oplus \Delta$

5. Break the correlation: $y^{j}_0' = H(y^{j}_0), y^{j}_1' = H(y^{j}_1)$

- **Not secure against active adversaries**
Part 2: Active Secure OT Extension

- Warmup: OT properties
- Recap: Passive Secure OT Extension
- Active Secure OT Extension
Active Security

1. Set each pair of messages $x^i_0, x^i_1$ s.t., $x^i_0 \oplus x^i_1 = \Gamma$

- How to force Bob to use same value?
- “Cut-and-choose”
  - Start with $\approx 2k$ OTs
  - Pair them at random (destroys half)
  - Check if the same $\Gamma$ was used
  - abort otherwise
The Equality BOX

- Output ok if equal
- abort/reveal all if different
The Equality BOX

\[ x \xrightarrow{\text{eq}} y \]

\[ H(x,r) \]

\[ x, r \xrightarrow{\text{ok/abort}} y \]
Pair and check

\[ u_1 = x_1 + b_1 \Gamma \]

\[ u_5 = x_5 + b_5 \Gamma \]

\[ d = b_1 + b_5 \]

\[ x_1 + x_5 + d \Gamma \]

EQ

OK

OK
Analysis

• Ok if both honest
  – \( u_i = x_i + b_i \Gamma_i \)
  – \( u_i + u_i = x_i + x_j + (b_i + b_j) \Gamma \) if \( \Gamma_i = \Gamma_j = \Gamma \)
  – Throw away OT \( j \) and keep \( i \) for later use

• Why use EQ?
  – Alice needs to prove \( d \) is correct too!
  – Else: corrupted Alice sends \( d = 1 + b_i + b_j \ldots \)
  – \ldots learns two MACs with same key
  – \ldots learns \( \Gamma \)
  – \ldots protocol breaks down completely
Corrupted Bob

\[ u_1 = x_1 + b_1 \Gamma + b_1 e_1 \]

\[ u_5 = x_5 + b_5 \Gamma + b_5 e_5 \]

\[ d = b_1 + b_5 \]

\[ x_1 + x_5 + d \Gamma + b_1 e_1 + b_5 e_5 \]

\[ u_1 + u_5 \]

\[ \text{ok} \]
Three cases

- **No error**: \( e_i = e_j = 0 \)
  - Bob always pass the check and learns nothing 😊

- **One error**: \( e_i \neq 0, e_j = 0 \)
  - Bob pass the test if guess \( b_i \) correctly
  - 50% abort, 50% Bob learns \( b_i \) 😞

- **Canceling errors**: \( e_i = e_j \neq 0 \)
  - Bob always pass the test
  - Can be simulated by leaking bit \( b_i \) 😞

For simplicity \( \forall i \ e_i \in \{0, e^*\} \)
Simulating

\[ u_1 = x_1 + b_1 \Gamma \]

\[ u_5 = x_5 + b_5 \Gamma \]

\[ d = b_1 + b_5 \]

\[ x_1 + x_5 + d \Gamma + d \epsilon \]

\[ \text{OK} \]

\[ \text{OK} \]
Simulating

\[ u_1 = x_1 + b_1 \Gamma \]

\[ u_5 = x_5 + b_5 \Gamma \]

\[ d = b_1 + b_5 \]

\[ x'_1 + x'_5 + d \Gamma \]

Where \( x'_i = x_i + b_i e \)
Three cases

• **No error**: \( e_i = e_j = 0 \)
  – Bob always pass the check and learns nothing 😊

• **One error**: \( e_i \neq 0, e_j = 0 \)
  – Bob pass the test if guess \( b_i \) correctly
  – 50% abort, 50% Bob learns \( b_i \) 😞

• **Canceling errors**: \( e_i = e_j \neq 0 \)
  – Bob always pass the test
  – Can be simulated by leaking bit \( b_i \) 😞

For simplicity \( \forall i \ e_i \in \{0, e^*\} \)
e = 0

- 0 + 0 = 0
  - No abort, no leak

- e + 0 ≠ 0
  - Abort with pr. ½, 1 bit leaked

- e + e = 0
  - No abort, 1 bit leaked

e = e*

- 2n

n
How many bits does Bob learn?

• **Define game:**
  – Choose how many \( e \neq 0 \). Abort \( \rightarrow \) loses
  – Receive \( b_i \) for all \( i \) in **yellow** and **red**
  – Guess *entire vector* \( b \). Wrong guess \( \rightarrow \) loses

• **Define leak** \( L < n + \log(\text{pr. Bob wins the game}) \)
  – Win = *not abort* + *correct guess*
  – \( \Pr(\text{not abort}) = 2^{-\#\text{yellow}} \)
  – \( \Pr(\text{correct guess}) = 2^{-\#\text{green}} \)

• \( L = n - \#\text{yellow} - \#\text{green} = \#\text{red} \)
e = e^* 

Optimal strategy

\[ n = \frac{4}{3k} \Rightarrow L < \frac{k}{3} \]
Finishing up...
OT Extension, Pictorially

1 - 2 OTs

k

n = poly(k)

U

4/3 k

1/3k

k

4/3 k

b

b

b

X₀

Γ

4/3 k
OT Extension, Pictorially

\[ \frac{4}{3}k \]

\[ \Delta \]

\[ Y_0 \]

\[ n = \text{poly}(k) \]

\[ 1 - 2 \text{ OTs} \]

\[ \text{Leak!} \]

\[ \frac{4}{3}k \]
Solutions

- **OT Extension:**
  - *Hash the leak away!*

- **Authenticated Bits (need linear relation)**
  - *Universal hash...*  
    (multiply with random matrix A)
  - *...or do nothing!*  
    (MAC still secure with $k$ unknown bits!)
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
    
    (where the adversary knows at most \(L\) bit).
  
  – MACs, keys are \(k\)-bit strings.
Authenticated Bits/OT Extension

1. Run \((2+2\mu)n\) OTs with constant difference \(\Gamma\)
2. \textit{Cut-and-choose} and throw away half OTs
3. \textit{Turn your head} (OT extension)

Authenticated Bits

4. \textit{Deal with \(\mu\)-leaked bits with universal hash (or don’t).}

OT Extension

4. \textit{Deal with \(\mu\)-leaked bits with cryptographic hash.}