Information:

- Presentation 25 minutes + 5 minutes for questions.

- Presentation is on Wednesday, 11:30-12:00 in B05-B06

- Presentation is after: Abhi Shelat
  (fast two-party secure computation with minimal assumptions)

- Presentation is before: Nigel Smart
  (An architecture for practical actively secure MPC with dishonest majority)

- BF Private Set-Intersection protocol is 2 sessions after us
More Efficient Oblivious Transfer and Extensions for Faster Secure Computation

Gilad Asharov
Yehuda Lindell
Cryptography Research Group
Bar-Ilan University

Thomas Schneider
Michael Zohner
Engineering Cryptographic Protocols Group
TU Darmstadt
1-out-of-2 Oblivious Transfer (OT)

- Input: Alice holds two strings \((x_0, x_1)\), Bob holds a choice bit \(r\).

- Output: Bob receives \(x_r\) but learns nothing about \(x_{1-r}\), Alice learns nothing about \(r\).
Motivation

- OT is basis of many generic secure computation protocols
  - Yao's garbled circuits protocol [Yao86]: one OT per input
  - Goldreich-Micali-Wigderson [GMW87]: one OT per AND gate

- Several special purpose protocols directly use OT:
  - Set-Intersection [DCW13]
  - Biometric identification [BCP13]

- We focus on semi-honest (passive) adversaries
  - Enables highly efficient protocols
OT via Public-Key Cryptography

- Several protocols for OT exist that use public-key cryptography
  - e.g., by [NP01] in random-oracle and standard model
  - Other protocols exist that require weaker security assumptions

- Impagliazzo and Rudich [IR86] proved that OT requires public-key cryptography

- Since public-key cryptography is expensive, OT was believed inefficient
OT Extensions

- OT extensions use secret-key cryptography to efficiently extend OT
  - OT on long strings by exchanging short seeds [Beaver96]
  - Many OTs extended from few “real” OTs [IKNP03]

- Similar to hybrid encryption, where symmetric key is encrypted using public-key cryptography
Our Contributions

- Optimizations for the OT extension protocol of [IKNP03]
  - Algorithmic optimizations => less computation
  - Protocol optimizations => less communication

- Specific OT functionalities for more efficient secure computation

- An open source OT extension implementation
OT Extension of [IKNP03] (1)

For each OT $i$:

- Alice holds $m$ pairs of $l$-bit messages $(x_{i,0}, x_{i,1})$

- Bob holds $m$-bit string $r$ and obtains $x_{i,r_i}$
OT Extension of [IKNP03] (2)

- Alice and Bob perform \( k \) “real” OTs on random seeds with reverse roles (\( k \) is symmetric security parameter)
OT Extension of [IKNP03] (3)

- Bob obliviously transfers a random $m \times k$ bit matrix $T$

- The matrix is masked with the seeds of the “real” OTs

\[
\begin{align*}
T & \in_R \{0, 1\}^{m \times k} \\
\text{for } 1 \leq j \leq k : & \\
& u_{j,0} = PRG(s_{j,0}) \oplus T[j] \\
& u_{j,1} = PRG(s_{j,1}) \oplus T[j] \oplus r \\
\text{for } 1 \leq j \leq k : & \\
V[j] & = u_{j,c_j} \oplus PRG(s_{j,c_j})
\end{align*}
\]
OT Extension of [IKNP03] (4)

- The $V$ and $T$ matrices are transposed
- Alice masks her inputs and obliviously sends them to Bob
  - $H$ is a correlation robust function (instantiated with a hash function)

\[ V' = V^T \quad T' = T^T \]

for $1 \leq i \leq m$:

\[
\begin{align*}
y_{i,0} &= x_{i,0} \oplus H(i, V'[i]) \\
y_{i,1} &= x_{i,1} \oplus H(i, V'[i] \oplus c)
\end{align*}
\]

for $1 \leq i \leq m$:

\[ x_{i,r_i} = y_{i,r_i} \oplus H(i, T'[i]) \]
Computation Complexity of OT Extension

Per OT:

1. # PRG evaluations
2. # H evaluations

Time distribution for 10 Mio. OTs (in 21s):

- "real" OTs: 1%
- PRG (AES-CTR): 33%
- Transpose: 42%
- Hash (SHA-1): 10%
- Misc (XOR/Snd/Rcv): 14%

For $m$ pairs $(x_{i,0}, x_{i,1}) \in \{0,1\}^{2l}$ and $r = (r_1, ..., r_m) \in \{0,1\}^m$:

- $c_j \in_R \{0,1\}$
- $s_{j,c_j} \leftarrow \text{OT}$
- $(s_{j,0}, s_{j,1}) \in_R \{0,1\}^{2k}$

For $1 \leq j \leq k$:

- $T \in_R \{0,1\}^{m \times k}$
- $T'[j] = T[j] \oplus r$

For $1 \leq j \leq k$:

- $u_{j,0} = \text{PRG}(s_{j,0}) \oplus T'[j]$
- $u_{j,1} = \text{PRG}(s_{j,1}) \oplus T'[j] \oplus r$

$V[j] = u_{j,c_j} \oplus \text{PRG}(s_{j,c_j})$

$V' = V^T$

$T' = T^T$

For $1 \leq i \leq m$:

- $y_{i,0} = x_{i,0} \oplus H(i, V'[i])$
- $y_{i,1} = x_{i,1} \oplus H(i, V'[i] \oplus c)$

$(y_{i,0}, y_{i,1}), 1 \leq i \leq m$ for $1 \leq i \leq m$:

$x_{i,r_i} = y_{i,r_i} \oplus H(i, T'[i])$
Algorithmic Optimization
Efficient Bit-Matrix Transposition

- Naive matrix transposition performs $mk$ load/process/store operations

- Eklundh's algorithm reduces number of operations to $O(m \log_2 k)$ swaps

- Use CPU register to swap multiple bit-values in parallel
  - $O(m/r \log_2 k)$ for register size $r$ (e.g., $r = 64$)

- Time for transposing the $m \times k$ bit matrix is reduced by factor 9
- OT extension can easily be parallelized by splitting the $T$ matrix into sub-matrices.

- Since each column is independent of the next, OT is highly parallelizable.
Communication Complexity of OT Extension

\[ m \text{ pairs } (x_{i,0}, x_{i,1}) \in \{0,1\}^{2l} \]
\[ r = (r_1, \ldots, r_m) \in \{0,1\}^m \]

- For \( 1 \leq j \leq k \):
  - \( c_j \in_R \{0,1\} \)
  - \( (s_{j,0}, s_{j,1}) \in_R \{0,1\}^{2k} \)

- For \( 1 \leq j \leq k \):
  - \( T \in_R \{0,1\}^{m \times k} \)
  - For \( 1 \leq j \leq k \):
    - \( u_{j,0} = PRG(s_{j,0}) \oplus T[j] \)
    - \( u_{j,1} = PRG(s_{j,1}) \oplus T[j] \oplus r \)

- For \( 1 \leq j \leq k \):
  - \( V[j] = u_{j,c_j} \oplus PRG(s_{j,c_j}) \)

\[ V' = V^T \]
\[ T' = T^T \]

- For \( 1 \leq i \leq m \):
  - \( y_{i,0} = x_{i,0} \oplus H(i, V'[i]) \)
  - \( y_{i,1} = x_{i,1} \oplus H(i, V'[i] \oplus c) \)

- For \( 1 \leq i \leq m \):
  - \( x_{i,r_i} = y_{i,r_i} \oplus H(i, T'[i]) \)

**Per OT:**
- 2l bits sent by Alice
- 2k bits sent by Bob

References:
- [Yao86]
- [GMW87]
- [BCP13]
Protocol Optimization
General OT Extension (G-OT)

- Instead of using a random $T$ matrix, we derive it from $s_{j,0}$:

- Reduces data Bob sends by factor 2

\[ T \in \mathbb{R} \{0,1\}^{m \times k} \]

for $1 \leq j \leq k$:

\[ u_{j,0} = PRG(s_{j,0}) \oplus T[j] \]

\[ u_{j,1} = PRG(s_{j,1}) \oplus T[j] \oplus r \]

for $1 \leq j \leq k$:

\[ \mathbf{V}[j] = u_{j,c_j} \oplus PRG(s_{j,c_j}) \]

for $1 \leq j \leq k$:

\[ u_j = PRG(s_{j,1}) \oplus T[j] \oplus r \]

\[ \mathbf{V}[j] = c_j u_j \oplus PRG(s_{j,c_j}) \]
Specific OT Functionalities

- Secure computation protocols often require a **specific OT functionality**
  - Yao's garbled circuits with free XOR [KS08] requires **correlated** inputs
  - GMW with multiplication triples can use **random** inputs

- We introduce two OT functionalities for secure computation protocols:
  - Correlated OT: **random** $x_0$ and $x_1 = x_0 \oplus \Delta$
  - Random OT: **random** $x_0$ and $x_1$
Specific OT Functionalities
Correlated OT Extension (C-OT)

- Choose $x_{i,0}$ as random output of $H$

- Compute $x_{i,1}$ as $x_{i,0} \oplus \Delta_i$ to obliviously transfer correlated values

- Reduces data Alice sends by factor 2
Specific OT Functionalities
Random OT Extension (R-OT)

- Choose $x_{i,0}$ and $x_{i,1}$ as random outputs of $H$

- Removes last communication step

\[
\begin{align*}
V' &= V^T \\
T' &= T^T \\
\text{for } 1 \leq i \leq m: & \quad y_{i,0} = x_{i,0} \oplus H(i, V'[i]) \\
& \quad y_{i,1} = x_{i,1} \oplus H(i, V'[i] \oplus c) \\
& \quad (y_{i,0}, y_{i,1}), 1 \leq i \leq m \\
& \quad \text{for } 1 \leq i \leq m: & \quad x_{i,r_i} = y_{i,r_i} \oplus H(i, T'[i])
\end{align*}
\]
Empirical Performance Evaluation

- Performance evaluation of 10 million OT extensions on 80-bit strings

- Two network types: Gigabit LAN and WiFi 802.11g
Empirical Performance Evaluation
Original Implementation

- C++ code of [SZ13] implementing OT extension of [IKNP03]
Empirical Performance Evaluation
Efficient Matrix Transposition

- Efficient matrix transposition => improved computation
- Only decreases runtime in LAN where computation is the bottleneck
Empirical Performance Evaluation
General Oblivious Transfer

- Generate $T$ from seeds => improved communication (Bob $\rightarrow$ Alice)

- WiFi runtime decreases only slightly, since communication Alice $\rightarrow$ Bob becomes the bottleneck
Empirical Performance Evaluation
Correlated Oblivious Transfer

- Correlated OT => improved communication (Alice → Bob)

- WiFi runtime decreases by factor 2
Empirical Performance Evaluation
Random Oblivious Transfer

- Random OT => improved communication (Alice → Bob)

- WiFi runtime does not decrease since communication Bob → Alice becomes the bottleneck
Empirical Performance Evaluation
Parallelized Oblivious Transfer

- Parallel OT extension with 2 and 4 threads => improved computation
- LAN runtime decreases linear in # of threads
- WiFi runtime remains the same (communication is the bottleneck)
Empirical Performance Evaluation

Conclusion

- LAN profits mostly from improved computation
- WiFi profits from improved communication
- Communication has become the bottleneck for OT extension
Summary

- **Communication** has become the **bottleneck** for OT

- New OT functionalities for more efficient secure computation
  - **Correlated OT** for correlated values
  - **Random OT** for random values

- Our OT implementation is available at [http://encrypto.de/code/OTExtension](http://encrypto.de/code/OTExtension)
  - A Java wrapper will be available in SCAPI
Thanks for your attention.

Questions?

Contact: http://encrypto.de
Protocol Overview

Special Purpose Protocols
- Homomorphic Encryption
- Public Key Crypto

Generic Protocols
- Arithmetic Circuit
- Boolean Circuit
- GMW
- Yao
- OT

Symmetric Crypto
- >>
- One-Time Pad
Generating Multiplication Triples via R-OT

- A multiplication triple has the form \((a_1 \oplus a_2) (b_1 \oplus b_2) = c_1 \oplus c_2\)

\[= (a_1 b_1) \oplus (a_1 b_2) \oplus (a_2 b_1) \oplus (a_2 b_2)\]

- \(P_1\) and \(P_2\) generate a multiplication using two R-OTs as follows:
  1) \(P_2\) chooses \(a_2 \in_r \{0,1\}\)
  2) \(P_1\) and \(P_2\) perform a random OT, where \(P_1\) gets \((x_1,x_2)\) and \(P_2\) gets \(x_{a_2}\)
  3) \(P_1\) computes \(b_1 = x_1 \oplus x_2\)
  4) \(P_1\) and \(P_2\) repeat steps 1-3 with reverse roles to get \(a_1\) and \(b_2\)
  5) \(P_i\) computes \(c_i = (a_i b_i) \oplus x_1 \oplus x_{a_i}\)
Efficient OT without Random Oracles

TODO:
Outline the protocol steps for the proposed base-OT