Secure Computation
EWSCS, Lecture 3

Claudio Orlandi
What we learned this morning

• If you can securely compute random multiplications, you can compute any function.

• This lecture: Oblivious Transfer (or how to compute random multiplications modulo 2)
  – Definition and simple reductions
  – OT Extension
  – OT from PKE+
  – NaorPinkas OT
  – PVW OT
What we learned this morning

How to compute \([z]=[xy]\) ?

Alice, Bob should compute

\[z_A + z_B = (x_A + x_B)(y_A + y_B)\]

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

Alice can compute this

Bob can compute this

How do we compute this?
On the use of computational assumptions

• How much can we ask users to trust crypto?
  1. **Necessary** (one way functions are needed for symmetric crypto, public key crypto is probably needed for 2PC)
  2. **We must believe that some problems are hard** (e.g., breaking RSA or breaking AES). But we should not ask for more trust than needed!
  3. Construct complex systems based on well studied assumptions. Then prove (via reduction), that *any adv that can break property X of system S can be used to solve computational problem P.*
  4. If we believe problem P to be hard, then we conclude that system S has property X.
The Crypto Toolbox

Weaker assumption

Stronger assumption

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

More efficient

Less efficient
• **If:** an adversary can break the security (e.g., learn the secret input $x$)

• **Then:** use this adversary as a subroutine to break the security of RSA

• **But:** we believe RSA is hard

• **So:** the protocol must be secure
Oblivious Transfer

- Receiver learns just one message
- Sender does not learn which one
- OT is a fundamental cryptographic primitive
  - Necessary and sufficient for MPC
  - Can be also instantiated using non crypto assumptions (noise, quantum)
The Crypto Toolbox

OT ≈ PKE

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

More efficient

Less efficient

Weaker assumption

Stronger assumption
Simple properties of Oblivious Transfer
$\text{OT} = \text{shared AND}$

$1-2 \text{ OT}$

Receiver

$\rightarrow b$

$\leftarrow x_b$

Sender

$\rightarrow x_0, x_1$

Bits
\[ OT = \text{shared AND} \]

\[ d = ab \oplus c \]

Bits

Receiver

Sender

1-2 OT
Stretching OT

Receiver

Sender

\[ (C_0, C_1) = (\text{Enc}(k_0, m_0), \text{Enc}(k_1, m_1)) \]

\[ m_b = \text{Dec}(k_b, C_b) \]
Random OT = OT

\[(C_0, C_1) = (r_0 \oplus m_0), (r_1 \oplus m_1)\]

\[m_b = r_b \oplus C_b\]

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

Receiver

\[ (C_0, C_1) = (r_0 \oplus d \oplus m_0), (r_1 \oplus d \oplus m_1) \]

\[ m_b = r_c \oplus C_b \]

\[ d = b \oplus c \]

Sender

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

Receiver

s₀, s₁

ROT

b, y = s₁

bits

Sender

r₀ = y

r₁ = b \oplus r₀

Exercise: check that it works

No communication!
OT Extension (passive security)
OT Extension

• OT provably requires public-key primitives.

• OT extension is the OT analogue of hybrid encryption
  – \((RSA(pk, k), AES(k, m))\)

• Start from \(k\) “real” OTs \(\rightarrow\) Extend them to \(\text{poly}(k)\) OTs using only few symmetric primitives (PRG/hash function) per generated OT
OT Extension, Pictorially

Remember: OT stretching
Condition for OT extension

\[ S_0 \oplus S_1 = \ldots = C \]

Problem for active security!
OT Extension, Pictorially

1-2 OTs

n=poly(k)
OT Extension, Pictorially

\[
Y = S0 \oplus b \otimes c
\]
OT Extension, Turn your head!

\[ Y = S_0 \oplus c = Z = R_0 \oplus c \]
OT Extension, Turn your head!

\[ Y = S_0 \oplus b = R_0 \oplus c = c \]
OT Extension, Pictorially

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

1-2 OTs

\[ \sigma \]

\[ \mathbb{Z} \]
Defining R1

\[ R1 = R0 \oplus b \]
Finishing UP

• R0, R1 are not really random (strongly correlated)

• **Compute** (for each of the n rows)
  – T0 = H(R0), T1 = H(R1)

• Using a **correlation robust hash function** H s.t.
  – \{X_0, \ldots, X_n, H(X_0 \oplus R), \ldots, H(X_n \oplus R)\}
  – \{X_0, \ldots, X_n, Y_0, \ldots, Y_n\}  // (X_i’s, Y_i’s random)

• Are computationally **indistinguishable**
OT Extension, Pictorially

\[ H(R_0), H(R_1) \]

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

\[ H(Z) \]

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

1. Start with $k$ 1-2 OT for strings of length $k$
2. **Stretch them** to length $n=\text{poly}(k)$ using a PRG
3. Set each pair of messages $s^{i}_{0}, s^{i}_{1}$ s.t., $s^{i}_{0} \oplus s^{i}_{1} = c$
4. Turn your head (s/r swap roles)
5. The bits of $c$ are the new **choice bits**
6. The new messages are of the form $r^{j}_{0}, r^{j}_{1} = r^{j}_{0} \oplus b$
7. Break the correlation: $t^{j}_{0} = H(r^{j}_{0}), t^{j}_{1} = H(r^{j}_{1})$
   • **Not secure against active adversaries**
How to construct OT
OT from PKE+

• Take a PKE (Gen, Enc, Dec)
  – Gen \(\rightarrow\) (pk, sk)
  – Enc(pk, m) \(\rightarrow\) C
  – Dec(sk, C) \(\rightarrow\) m
  – “IND-CPA Security”
    \((pk, Enc(pk, m)) \sim (pk, Enc(pk, r))\)

• Augmenetted with (OGen, OGen\(^{-1}\))
  – OGen(r) \(\rightarrow\) pk
  – OGen(pk) \(\rightarrow\) r

\[ pk^{OGen} \sim pk^{Gen} \]
The idea: it should be possible to generate “good looking” public keys without learning the secret key.
OT from PKE+

- **Receiver** (with choice bit b)
  - Compute $pk_b, sk \leftarrow \text{Gen}$
  - Compute $pk_{1-b} \leftarrow \text{Ogen}$
  - Send $pk_0, pk_1$
- **Sender** (with messages $m_0, m_1$)
  - Encrypt $C_0 = E(pk_0, m_0), C_1 = E(pk_1, m_1)$
- **Receiver**
  - $m_b = \text{Dec}(sk, C_b)$

- **What can go wrong?**
Discrete Log (additive notation)

- \(<G>\) group of order \(p\) (prime) generated by \(G\)
  - DL: given \(H\) it is hard to compute \(x\) s.t. \(xG = H\)
  - CDH: given \((G, aG, bG)\) it is hard to compute \(abG\)
  - DDH: given \((G, aG, bG, cG)\) it is hard to decide if \(c = ab\)
- ElGamal Encryption
  - Gen \(\rightarrow (G, H = xG)\)
  - Enc\((pk, m) \rightarrow (rG, rH + m)\)
  - Dec\((sk, C, D) \rightarrow D - xC\)
- In “many” groups it is possible to sample uniform \(H\)
  (without learning DL)
Exercise: we do not know how to sample RSA public keys obliviously. But ciphertexts can. Can you build an OT protocol from RSA?
Naor Pinkas OT

• Sender
  – Choose random F

• Receiver (with choice bit b)
  – Pick $H_b = xG$
  – $H_{1-b} = F - H_b$
  – Send $H_0$

• Sender (with messages $m_0, m_1$)
  – Compute $H_1 = F - H_0$
  – Encrypt $Enc(H_0, m_0), Enc(H_1, m_1)$

  //idea: hard to compute DL of both $H_0, H_1$
  (but cannot simulate)