Secure Hamming Distance based Computation and its applications

Ayman Jarrous, Benny Pinkas
University of Haifa

In this talk

- Secure two-party computation of functions which depend only on the Hamming distance of the inputs.
- Security against malicious adversaries, according to the notion of full simulatability.
  - Therefore when used as a building block for more complex protocols, these can be analyzed assuming building block implemented by a trusted party.
- Applications (*m*-point-SPIR).

HDOT – Hamming Distance based Oblivious Transfer

- Hamming distance $d_H(w,w')$

- HDOT – Hamming Distance based Oblivious Transfer, is a secure two-party protocol, where
  - P_1 has input w, P_2 has input w'. ($|w|=|w'|=m$)
  - P_1 learns an output which depends only on $d_H(w,w')$

- More specifically,
  - P_2 has additional inputs $Z_0,\ldots,Z_m$
  - P_1 learns Z_d, where $d = d_H(w,w')$.

HDOT- Straightforward applications

- Computing the Hamming distance ($Z_i=i$)
- Computing scalar product ($Z_i = i \bmod 2$)
- EQ – Equality based transfer: $EQ_{V0,V1}(w,w')$ outputs V_0 if $w=w'$, and V_1 otherwise.
  - Set $Z_0=V_0$, $Z_1=\ldots=Z_m=V_1$.
  - There is a specific output V_1 if $w\neq w'$. This is harder than outputting a random output in that case.
- Threshold HDOT: output V_0 if $d_H(w,w')<t$, and V_1 otherwise.
- Computing similarity of documents.

Trivial implementations

- Recall, *1*-out-of-*N* oblivious transfer
  - P_2 has input $X_1,\ldots,X_N$. P_1 has input $1 \le i \le N$.
  - P_1 learns X_i.
  - Can be implemented using logN 1-out-of-2 OTs.
- HDOT using 1-out-of-2^m OT…
- HDOT using generic two-party computation (Yao)
  - Requires *m* 1-out-of-2 OTs, and a circuit for $d_H(w,w')$.

- Secure protocols computing scalar product, based on hom. enc. [WY,GLLM], semi-honest security.
- Approximation protocols [IW]. Good asymptotically.

Our results

- Recall, *1*-out-of-*N* oblivious transfer
  - P_2 has input $X_1,\ldots,X_N$. P_1 has input $1 \le i \le N$.
  - P_1 learns X_i.
  - Can be implemented using logN 1-out-of-2 OTs.
- HDOT using 1-out-of-2^m OT…
- HDOT using generic two-party computation (Yao)
  - Requires m 1-out-of-2 OTs, and a circuit for $d_H(w,w')$.

- Our results:
  - log(*m*) OTs, *m* homomorphic encryptions.
  - Security against malicious adversaries.

The basic protocol

- *bin*HDOT. *w,w'* are binary strings.

- First step: Use homomorphic encryption to let P_1 learn $d_H(w,w')+R$, computed in a large field or ring.

- Second step: Run *1*-out-of-*(m+1)* OT to let P_1 learn desired protocol output.

*bin*HDOT – First step

- $w=w_0,\ldots,w_{m-1}$; $w'=w'_0,\ldots,w'_{m-1}$
- P_1 knows the decryption key of a homomorphic encryption scheme over $\mathbb{F}$. ($|\mathbb{F}| \gg m$)

- P_1 sends $E(w_0),\ldots,E(w_{m-1})$
- P_2 uses homomorphic properties
  - To compute $E(w_0 \oplus w'_0),\ldots,E(w_{m-1} \oplus w'_{m-1})$
  - To sum these values and obtain $E(d_H(w,w')) = E(d)$
- P_2 chooses random $R \in \mathbb{F}$ and sends $E(d+R)$ to P_1

*bin*HDOT – Second step

- P_1 decrypts $E(d+R)$, and reduces the result modulo $m+1$.
  - Note that $d \leq m \ll |\mathbb{F}|$, and therefore whp computing $d+R$ in $\mathbb{F}$ involves no modular operation.
  - Therefore the decrypted value equals $d+R$ computed over the integers, and P_1 computes $d+R \bmod m+1$.
- The parties run a 1-out-of-$(m+1)$ OT
  - P_1's input is $d+R \bmod m+1$
  - P_2's rotates the order of $Z_0,\ldots Z_m$ by R, and sets its input to be $Z_{-R \bmod (m+1)}, Z_{-R+1 \bmod (m+1)},\ldots,Z_{-R+m \bmod (m+1)}$
  - P_1 learns $Z_{(d+R)-R \bmod (m+1)} = Z_d$

Overhead

- Recall, overheads
  - OT > homomorphic encryption > homo. addition
- Overhead
  - Our protocol: *m* encryptions/additions, log*m* OTs.
  - Yao: *m* OTs + evaluating a circuit.
- Precomputation:
  - Our protocol: P_1 can precompute all *m* encryptions without communicating with P_2.
  - Yao: Precomputing the OTs requires interaction.

Security against malicious adversaries

- In the sense of full simulatability
  - I.e., the parties are aware of their inputs.
  - Anything an adversary does in the real protocol, it can also do in an ideal scenario including a TTP.

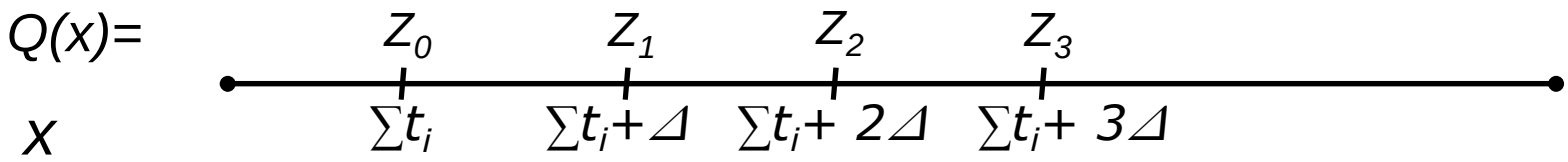
- Some issues:
  - Ensure that in the OT, P_1 can only learn item $d_H(w,w')$.
  - Make sure that P_2 "knows" its inputs to the OT.

Security against malicious adversaries: Tools

- Committed OT with constant difference (COTCD)
  - P_2 has inputs m_0, m_1, s.t. $|m_0-m_1|=\Delta$.
  - P_1 has inputs $i \in \{0,1\}$, com(m_0), com(m_1), com(Δ).
  - Output of P_1: output m_i, but only if $|m_0-m_1|=\Delta$
  - P_1 does not learn Δ, but verifies that $|m_0-m_1|= \Delta$
  - Implementation based on JS committed OT.
- Oblivious Polynomial Evaluation (OPE)
  - P_2 has polynomial Q(). P_1 has X. P_1 learns Q(x).
  - We use a construction of Hazay-Lindell.

Security against malicious adversaries: The protocol

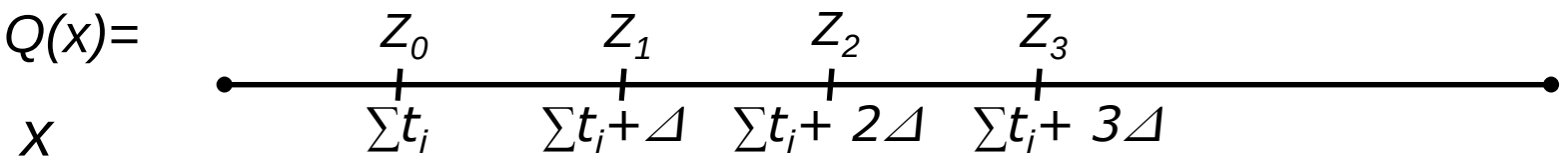
- P_2 chooses a random Δ
- For every letter location $0 \leq i \leq m-1$, run a COTCD, where
  - P_2 chooses a random value t_i
  - P_1 learns t_i if $w_i = w'_i$. Learns $t_i + \Delta$ otherwise.
- P_1 sums the values it received: gets $\sigma = \sum t_i + \Delta d_h(w,w')$
- P_2 prepares a polynomial $Q()$, s.t. $\forall j \;\; Q(\sum t_i + \Delta \cdot j) = Z_j$.



Security against malicious adversaries: The protocol

- The parties run an OPE where P_1 learns
  $Q(\sigma) = Q(\sum t_i + \Delta d_h(w,w')) = Z_{d(w,w')}$

- P_1 does not know Δ. Therefore if it asks to learn a different value of $Q()$, whp it does not hit any of $\sum t_i$, $\sum t_i + \Delta$, $\sum t_i + 2\Delta$, …



An application: *m*-point SPIR

- SPIR – Symmetric PIR
  - P_2 has inputs $X_0,\ldots,X_{N-1}$
  - P_1 has input $i \in 0,\ldots,N-1$
  - P_1 learns X_i in $o(N)$ communication. Nothing else leaks.
- *m*-point SPIR
  - At most *m* inputs of P_2 have unique values. All other inputs have a default value α.
  - P_1 must not know whether it learns the default value α or one of the *m* unique values.
  - (If we allow P_1 to learn a *random* value instead of the default value α, then easy to implement, using OPE.)

1-point SPIR

- 1-point SPIR
  - X_{i1} is unique. For all other i, $X_i=\alpha$.
  - If P_1's input i is equal to i_1, P_1 must output X_{i1}, otherwise it outputs α.
  - I.e., if $d_H(i,i_1)=0$, P_1 must output X_{i1}…
  - Implement using $EQ_{Xi1,\alpha}(i,i_1)$. I.e., P_1 learns X_{i1} if its input i is equal to i_1. Learns α otherwise.
  - (The EQ protocol is an invocation of HDOT.)
  - log$N$ communication, loglog$N$ OTs, log$N$ encryptions.

m-point SPIR

- *3*-point SPIR (*m*-point SPIR is identical)
  - X_{i1} , X_{i2} , X_{i3} are unique. For all other i, $X_i=\alpha$.
  - If P_1's input i is in $\{i_1,i_2,i_3\}$, then P_1 must output X_i, otherwise it outputs α.

Protocol for *3*-point SPIR

- P_2 chooses random z'_1,z'_2,z'_3 s.t. $z'_1 \oplus z'_2 \oplus z'_3 = \alpha$
- Sets z_1, s.t. $z_1 \oplus z'_2 \oplus z'_3 = X_{i1}$
- Sets z_2, s.t. $z'_1 \oplus z_2 \oplus z'_3 = X_{i2}$
- Sets z_3, s.t. $z'_1 \oplus z'_2 \oplus z_3 = X_{i3}$

- The parties run $EQ_{Z1,Z'1}(i,i_1)$, $EQ_{Z1,Z'2}(i,i_2)$, $EQ_{Z1,Z'3}(i,i_3)$.
- P_1 xors the three results it obtains.

- If $i \neq i_1,i_2,i_3$, then P_1 learns z'_1,z'_2,z'_3, whose xor is α.
- If $i = i_1$, then P_1 learns z_1,z'_2,z'_3, whose xor is X_1. Etc…

m-point SPIR

- *m*-point SPIR requires *m* invocations of HDOT.
  - Communication is $O(m \log N)$.
- If the number *m* of unique points is $o(N / \log N)$, then communication is $o(N)$, as required in SPIR.
- *m*-point SPIR can be based on OT alone (unlike PIR).

Conclusions

- Focus on a class of useful functions.
- Find efficient protocols.
- Secure against malicious advesaries.
- A new variant of SPIR.