CRYPTOGRAPHIC PROTOCOLS
2016, LECTURE 7

SECURE COMPUTATION WITH BDD

Helger Lipmaa
University of Tartu, Estonia
UP TO NOW

- Introduction to the field
- Simple secure computation protocols
  - based on additively homomorphic encryption
- Common limitation: class of affine functionalities
  - although various tricks increase the class slightly
THIS TIME

- Damgård-Jurik cryptosystem
  - rate-optimal additively homomorphmic encryption
- Secure computation of a wide class of protocols
  - recursion, Binary Decision Diagrams
- Includes my own research (2005, 2009, ...)
- 20 minutes overtime in 2014/2015
REMINDER: TRAPDOOR DL

Def. Binomial coefficient:
\[
\binom{n}{i} = \frac{n!}{i!(n-i)!}
\]

Relation with exponent:
\[
(a + b)^m = \sum_{i=0}^{m} \binom{m}{i} a^i b^{m-i}
\]

Need a, b:
\[
a = n, \ b = 1:
(n + 1)^m = \sum_{i=0}^{m} \binom{m}{i} n^i = \binom{m}{0} n + \binom{m}{1} + \binom{m}{2} n^2 + ...
\]

modulo n^2:
\[
(n + 1)^m \equiv 1 + mn \mod n^2
\]

Idea: while encrypting, use \( g = n + 1 \) as a generator in a group modulo \( n^2 \). Needed: can compute DL only while knowing some secret.
LARGER MODULI

✧ **Paillier 1999**: TDL modulo $n^2$ for $m \in \mathbb{Z}_n$

✧ **Damgård-Jurik 2001**: TDL modulo $n^{s+1}$, $s \geq 1$, for $m \in \mathbb{Z}_{n^s}$

- $\phi (n^{s+1}) = n^{s+1} \cdot (1 - 1 / p) (1 - 1 / q) = \phi(n) n^s$

- $(r^{ns})^\phi = r^{\phi(n^{s+1})} \equiv 1 \mod n^{s+1}$ \hspace{1cm} // denote $\phi = \phi(n)$

✧ **Encryption**: $\text{Enc}^s (m; r) := (n + 1)^m r^{ns} \mod n^{s+1}$

✧ **Decryption**: recover $m \mod n$ as in Paillier, then $m \mod n^2$, etc
1. Compute \( c^\phi = (1 + n)^{m\phi} = 1 + m\phi n + \binom{m\phi}{2} \cdot n^2 \mod n^3 \)

2. Obtain \( m_0 = m \mod n \) from \( c^\phi = 1 + m\phi n \mod n^2 \) (as in Paillier)

3. Write \( m = m_0 + m_1 n \), for yet unknown \( m_1 \), where \( m_i \leq n \)

4. Thus \( c^\phi = 1 + (m_0 + m_1 n)\phi \cdot n + \binom{m_0 + m_1 n\phi}{2} \cdot n^2 \)

   \[ = 1 + m_0\phi \cdot n + \left( m_1\phi + \binom{m_0\phi}{2} \right) n^2 \mod n^3 \]

   \[ \Rightarrow d \leftarrow \frac{c^\phi - \left( 1 + m_0\phi \cdot n + \frac{m_0\phi}{2} \cdot n^2 \right)}{n^2} \mod n^3 = m_1\phi \mod n \]

7. Compute \( m_1 \leftarrow d\mu \mod n \)

   Recall \( \mu = \phi^{-1} \mod n \)

   Recall \( n^2 \) is not invertible \( \mod n^3 \)
DAMGARD-JURIK ENCRYPTION

**DJ.Setup** ($\mathcal{r}$):
1. Choose good keylength (length of $p$, $q$)
2. Return $gk \leftarrow$ keylength

**DJ.Keygen** ($gk$):
1. $p$, $q \leftarrow$ random keylength-long primes
2. $n \leftarrow pq$
3. $\phi \leftarrow (p - 1)(q - 1); \mu \leftarrow \phi^{-1} \mod n$
4. Return $((\phi, \mu), n)$ /* secret key, public key */

**DJ.Enc** ($gk$, $n$; $m$, $r$):
1. /* Assumes $m \in \mathbb{Z}_n^s$ for $s \geq 1$ */
2. /* Assumes $r \leftarrow \mathbb{Z}_n^*$: randomized alg. */
3. $c \leftarrow (n + 1)^m r^n \mod n^{s+1}$
4. Return $c$

**DJ.Dec** ($gk$, $\phi$, $\mu$; $c$):
1. $c^* \leftarrow c^\phi \mod n^{s+1}$
2. Iterative decryption:
   1. recover $m \mod n$ as in Paillier
   2. then $m \mod n^2$ etc
3. Return $m$
Theorem. DCR is \((\approx \tau, \approx \varepsilon)\)-secure iff DJ is \((\tau, \varepsilon)\)-IND-CPA secure.

Proof idea. Similar to Paillier but need to take care about reducing the case \(s \geq 2\) to \(s = 1\). (Will not prove.)
**DJ: PROPERTIES**

- **Additive homomorphism:** (exactly as in Paillier)
  
  \[ \text{Enc}^s(m; r) \text{Enc}^s(m'; r') = \text{Enc}^s(m + m'; rr') \]

- **Good rate:**
  
  \[ |m| = s \log n, \quad c := |\text{Enc}^s(m; r)| = \log n^s + 1 = (s + 1) \log n \]

  \[ \text{rate} := \frac{|m|}{|c|} = \frac{s}{(s + 1)} = 1 - \frac{1}{(s + 1)} \rightarrow 1 \]

One of the very few known cryptosystems that has both properties

(Elgamal is homomorphic but small rate, AES-CBC has good rate)
Let $c = \text{Enc}^{s+1}(m; r) = (n + 1)^m r^{n^{s+1}} \mod n^{s+2}$ for $m \in \mathbb{Z}_{n^s}$.

Since $m \in \mathbb{Z}_{n^s}$ then $c' = \text{Enc}^s(m; r)$ decrypts to $m$.

In some protocols we need both $c'$ and $c$.

Compression:

$c' = c \mod n^{s+1} \Rightarrow$ thus one only has to transfer $c$. 
Factoring of $n$ must be hard

Thus $|n| \geq 2000$

$|n| (s + 1) = 2000 (s + 1)$-bit arithmetic

Gets **extremely slow** for large $s$

**Tradeoff:** rate 1 but slow
(2, 1)-CPIR$^s$ WITH DJ

$s \leftarrow \text{Ceil}(L/\log n)$

$r \leftarrow \mathbb{Z}_n^*$

$c \leftarrow \text{Enc}^s(x; r)$

$c' \leftarrow \mathbb{Z}_n^*$

$\text{Reply} \leftarrow c f_i - f_0 \cdot \text{Enc}^s(f_0; r')$

$fx \leftarrow \text{Dec}^s(\text{Reply})$

$CPIR^s(x) := \text{Reply}$

Almost exactly like with Paillier

$f_x \in \mathbb{Z}_n^s$ $\Rightarrow$ Reply $= \text{Enc}^s(f_x) \in \mathbb{Z}_n^{s+1}$

pk, sk

$L$

$x \in \{0, 1\}$

pk

$f = (f_0, f_1)$

$f_i \in \{0, 1\}^L$
Computer programs consist of primitive operations

Every operation by itself: usually not so powerful

not so much you can do with only addition, or only if/while commands

Several op-s together can give a lot of power

Same here

homomorphic encryption + recursion => power
Clearly, correct plaintext / ciphertext sets

\[ m_0 \in \mathbb{Z}_{n^s} \]

\[ m_1 = \text{Enc}^{s}(m_0) \in \mathbb{Z}_{n^{s+1}} \]

\[ m_2 = \text{Enc}^{s+1}(\text{Enc}^{s}(m_0)) \in \mathbb{Z}_{n^{s+2}} \]

\[ m_i = \text{Enc}^{s+i-1}(\ldots (\text{Enc}^{s}(m_0)) \ldots) \in \mathbb{Z}_{n^{s+i}} \]
(4,1)-CPIR WITH RECURSION

\[ f_{0,0} \in \mathbb{Z}_{n^s} \]
\[ f_{0,1} \in \mathbb{Z}_{n^s} \]
\[ f_{1,0} \in \mathbb{Z}_{n^s} \]
\[ f_{1,1} \in \mathbb{Z}_{n^s} \]

CPIR^s (x_0)

\[ f_{2,0} := \text{Enc}^s (f_{0,x_0}) \in \mathbb{Z}_{n^s + 1} \]

CPIR^s (x_0)

\[ f_{2,1} := \text{Enc}^s (f_{1,x_0}) \in \mathbb{Z}_{n^s + 1} \]

CPIR^{s+1} (x_1)

\[ f_{3,0} = \text{Enc}^{s+1} (f_{2,x_1}) = \text{Enc}^{s+1} (\text{Enc}^s (f_{x_1,x_0})) \in \mathbb{Z}_{n^{s+2}} \]
(4, 1)-CPIR$^S$ WITH DJ

\[ s \leftarrow \text{Ceiling}(L/\log n) \]
\[ r_0, r_1 \leftarrow \mathbb{Z}_n^* \]
\[ c_0 \leftarrow \text{Enc}^s(x_0; r_0) \]
\[ c_1 \leftarrow \text{Enc}^s+1(x_1; r_1) \]
\[ r_1', r_2' \leftarrow \mathbb{Z}_n^* \]
\[ f_{20} \leftarrow \text{c}_0 \cdot f_{01} \cdot \text{Enc}^s(f_{00}; r_1') \]
\[ f_{21} \leftarrow \text{c}_0 \cdot f_{10} \cdot \text{Enc}^s(f_{10}; r_1') \]
\[ \text{Reply} \leftarrow c_1 \cdot f_{21} \cdot f_{20} \cdot \text{Enc}^{s+1}(f_{20}; r_2') \]

\[ M \leftarrow \text{Dec}^{s+1}(\text{Dec}^s(\text{Reply})) \]

\[ M = \text{Dec}^{s+1}(\text{Dec}^s(\text{Reply})) = \text{Dec}^{s+1}(f_{2, x_1}) = f_{x_1, x_0} = f_x \]
EFFICIENCY: (4, 1)-CPIRS

- **Communication:**
  - $2 + 1 = 3$ ciphertexts of length $\leq (s + 2) \log n$

- **Alice's computation:**
  - two encryptions, 2 decryptions (variable length)

- **Bob's computation:**
  - 3 encryptions, 3 exp-s (variable length)
SIMPLER PICTURE

Communication: 
\((\#x\text{-variables} + 1) \times (s + \text{length})\) \(\kappa\) 
\(* 3 \times (s + 2)\) \(\kappa\) *

Bob computation: 
\(\Theta(\text{size})\) exp-ns, each has 
\(\leq (s + \text{length})\)-bit inputs
DECISION TREE
DECISION TREE
Arbitrary **directed acyclic graph** $G$

- **Sinks** labeled by values $f_i$
  - **Large-output BDD:** $f_i$ any integers
- **Non-sink nodes** labeled by variables $x_i$
- **Single source**
1. $v = \text{source}$

2. While $v->\text{is_sink} = \text{false}$
   
   1. If $x[v] = 0$:  $v = v->\text{left_successor}$
   
   2. else:  $v = v->\text{right_successor}$

3. Return $f[v]$
BDD: COMPUT. MODEL

- BDD is a general **computation model**
  - can define a BDD that computes *any* Boolean function $f: \{0,1\}^L \rightarrow \{0,1\}$
  - non-Boolean case is a straightforward extension
  - but it is not always an **efficient** model
  - for many functions, arithmetic circuit is much smaller than BDD
BDD EVALUATION COMPLEXITY

- **Space complexity:** $\Theta(size \cdot \log(size))$
  - every node and edge of BDD has to be stored
- **Time complexity:** $\leq$ length
  - only one path has to be followed at every time
- **Preprocessing complexity:** $\Omega(size \cdot \log(size))$
  - and might be much larger

Computing optimal BDD-s is NP-hard
Poly-size BDD exist for the class of languages computable by logarithmic-space Turing machines with advice

- class **L/poly** [Cobham 1966]
- Translation: most of the interesting efficient (poly-time) functions
- ...but not all!
BDD: DATA STRUCTURE

- Often used as a data structure (to represent data)
  - Storing/preprocessing expensive but searching fast
- Donald Knuth called BDDs "one of the only really fundamental data structures that came out in the last twenty-five years"
- In certain applications, ubiquitous
  - circuit testing, ...
BDD: INFORMATION RETRIEVAL

- Fetch $f_x$, where $x = \sum x_i \cdot 2^i$
- Example with $N = 3$ // $x = x_0 + 2x_1 + 4x_2$
- **Length:** $N$
- **Size:** $1 + 2 + \ldots + 2^{N-1} = 2^N - 1$
  - Linear in database size $2^N$
- **Size** can be optimised
  - **Optimal:** $\Theta(2^N / N)$ [Lipmaa 2009]

Any function $f: \{0,1\}^N \rightarrow \{0,1\}$ can be computed with BDD of size $\Theta(2^N / N)$ --- known from 1990s
"Millionaire's" \hspace{1cm} // \textit{b} known by Bob

Example: \[a = 5\]

Three variables: \(x_0, x_1, x_2\)

Is \(4x_2 + 2x_1 + x_0 = 5\)?

Length: \(\log a\)

Size: \(\log a\)

Idea: \(5 = 101\) in binary
(QUIZ) BDD: \([A < B]\)

• Example: \([a < 5]\)
  • Three variables: \(x_0, x_1, x_2\)
  • Is \(4x_2 + 2x_1 + x_0 < 5?\)

• Length: \(\log a\)
• Size: \(\log a\)

Idea: \(5 = 101\) in binary
(QUIZ) BDD: MAJORITY

• \( \text{maj}(x_1, ..., x_{2N+1}) := (\sum_i x_i > N \ ? \ 1 : 0) \)

• Example with \( N = 1 \)  // \( (x_0, x_1, x_2) \)
(QUIZ) BDD: MAJORITY

- \( \text{maj}(x_0, ..., x_{2N}) := (\Sigma_i x_i > N ? 1 : 0) \)
- Example with \( N = 1 \) // \((x_0, x_1, x_2)\)
- **Length:** \( N \)
- **Size:** \( 1 + 2 + ... + N + (N + 1) + N + ... + 2 + 1 = (N + 1)^2 \)
- Size can be optimised for large \( N \)
LEVELLED BDD

- We need that every node has well-defined level
- \( \text{length (source)} = \text{length (bdd)} \)
- If exists edge \( i \rightarrow j \)
  then \( \text{length (} j \text{)} = \text{length (} i \text{)} + 1 \)

All paths have equal length => PrivateBDD will be well defined
BACK TO SECURE COMPUTATION

- \((4,1)\)-CPIR protocol: private implementation of BDD for information retrieval

- One can analogously implement any BDD

- However: privacy costs

  - If Bob does not do any computation at some node of BDD, he "knows" that the computation never reached this node

  - Thus time complexity: \(\Omega\) (size)
1. /* Order nodes so that there is edge i->j only if i < j */

2. /* Assume c[j] encrypts x[j] */

3. for (v = #sinks + 1; v ≤ size; v++)
   1. i = f[R[v]] - f[L[v]]
   2. s = Ceiling (L / log n) + length[v] - 1
   3. f[v] = c[x[v]]^i * Enc_s(f[L[v]]); ...

4. Return f[size]
PRIVATEBDD

1. $d \leftarrow \text{length (BDD)}$
2. for $i = 0$ to $t$
   1. $r_i \leftarrow \mathbb{Z}_n^*$
   2. $c_i \leftarrow \text{Enc}^{s+d-1}(x_i, r_i)$

$c[0], ..., c[t]$
CORRECTNESS

- Follows from the construction and correctness of
  - BDD and
  - CPIR at every node
隐私

- **Alice's privacy:**
  - Bob only sees random ciphertexts
  - thus, follows from DCR assumption

- **Bob's privacy:**
  - Alice sees a multiple-encryption of intended output by using length-dependent randomisers
  - each randomiser is different => each intermediate ciphertext random
EFFICIENCY

- **Communication:**
  - \((\#x\text{-var} + 1)\) ciphertexts, \((s + \text{length}) \cdot \log n\) bits each

- **Alice's computation:**
  - \(#x\text{-var}\) encryptions, \text{length} decryptions

- **Bob's computation:**
  - \text{size} encryptions and exponentiations
EFFICIENCY: $A < B$?

- Bob precomputes BDD for his $b$, write $N = \log a$
- Let $m := (s + d) \log n = (s + N) \log n$
- Communication:
  - $N + 1$ ciphertexts, each $m$ bits
- Alice's computation:
  - $N$ encr and decr with $m$-bit modulus
- Bob's computation:
  - $N$ exponentiations and encryptions
COMPUTATION: $(2^N, 1)$-CPIR

- Alice's computation:
  - $N$ encryptions and 1 decryption
  - with $\approx (S+N)$-bit modulus

- Bob's computation:
  - $2^N - 1$ exponentiations and encryptions
  - with $\approx (S+N)$-bit modulus
  - superexpensive
## COMMUNICATION: $(2^N, 1)$-CPIR

### Bob:
- 1 ciphertext of length $s + N$
- $c_B := (s + N) \kappa = L + N\kappa$ // approx.

### Alice:
- 1 ciphertext of length $d = s + 1, s + 2, ..., s + N$
  - $m_d := (s + d)\kappa = L + d\kappa$ bits
  - $c_A := LN + N(N + 1)\kappa / 2$
- **Total:** $c_{AB} := (N + 1)L + N(N + 3)\kappa / 2$
- **Rate:** $1 / (N + 1)$
(2^N, 1)-CPIR: BETTER COMMUNICATION

✧ **“Bitwise" optimization** (again):
  ✧ divide database into \( t \) pieces
  ✧ run \( t \) protocols in parallel with \( (L / t) \)-bit databases

✧ \( c_A(t) = L / t \cdot N + N (N + 1) \kappa / 2 \) // Alice’s message reused

✧ \( c_B(t) = (L / t + N \kappa) \cdot t = L + N \kappa / t \)

✧ \( c(t) = (1 + N / t) L + N (N + 1) \kappa / 2 + N \kappa / t \)

✧ Choose \( t = t_0 = (2 L / \kappa)^{1/2} \): // near-optimal, unpublished

✧ \( c(t_0) = (1 + N / t_0) L + ... = L + O (L^{1/2}) \) —— rate \( 1 - o(1) \)
SHORT HISTORY OF COIR

- [Lipmaa 2005]:
  - First CPIR with "good" communication
  - Rate: $\frac{1}{N + 1}$
- [Ishai Paskin 2007]:
  - Generalised to any BDD
- [Lipmaa 2009]:
  - Due to upper bound on BDD size, can do CPIR in sublinear time
  - Rate: $\frac{1}{2}$
- [Kiayias, Lipmaa, Leonardos, Pavlyk, Tang, 2015]:
  - Rate: 1
STUDY OUTCOMES

- **DJ**: rate-optimal homomorphic encryption
- **Recursion**
- **BDD**
- **Private BDD**
- Concrete protocols: \((2^N, 1)\)-CPIR, millionaire's, ...

Much more interesting than before
WHAT NEXT?

- We saw how to use homomorphism to obtain interesting protocols

- **Next lecture:**
  - one more “tool”: **multiple round protocols**
  - can do (say) multiplications efficiently
  - + a bit of **multi-party** computation