CS 6262 - Network Security: Secure Multiparty Computation

Hank Carter
Professor Patrick Traynor
4/25/13
Secure Computation

• Goal: to compute a shared result between untrusting parties.

• Example: Millionaire’s problem

• Simple solution: Trusted Third Party

• Can we get the same security based on cryptographic assumptions instead?
What's the point?
How do we do it?

• Many different techniques (e.g., ORAM, Secret-Sharing)
• Some special-purpose protocols used in practice
  ‣ Partially Homomorphic Encryption (e.g., ElGamal)
• Two primary techniques for general computation:
  ‣ Fully Homomorphic Encryption
  ‣ Garbled Circuits
What’s the difference?

• Fully Homomorphic Encryption:
  ‣ Arithmetic circuits
  ‣ Non-interactive protocols (good)
  ‣ Extreme computational costs (bad)

• Garbled Circuits
  ‣ Logical circuits
  ‣ Online interactive phase required (bad)
  ‣ Uses symmetric key operations and hashing (mostly) (good)
Garbled Circuits

• Developed by Andrew Yao in 1982
  ‣ First construction for SMC

• Basic concept:
  ‣ Turn a function into a logical circuit
  ‣ Turn logical gates into “garbled” truth tables
  ‣ Based on the input wire “label”, decrypt a single output wire “label”

• Necessary interactivity: one party constructs the circuit while the other evaluates it
An example gate garbling and evaluation

- Exclusive-OR
- Steps to garble:
  - Build truth table
  - Choose wire labels
  - Encrypt output wire labels
  - Permute
Oblivious Transfer

- 1-out-of-2 OT:
  - One party offers two items
  - The other party chooses one
  - The first party doesn’t know which item was chosen, while the second party doesn’t know what the other item is

- Necessary for exchanging input wire labels
- Typically require expensive crypto
A simple OT example

- An RSA-based scheme by Even, Goldreich, and Lempel (See Wikipedia article on “Oblivious Transfer”)

- Steps for 1-out-of-2 OT:
  - Sender chooses two random messages
  - Chooser picks one and blinds it with a random encryption
  - Sender removes both random messages and decrypts to create two blinds for both selection messages
  - Chooser removes the only random blind he possesses
The Fairplay Implementation

• Compiling the logical circuit:
  ‣ SFDL => SHDL
  ‣ Security requirements (e.g., loop unrolling)
  ‣ Compiler optimizations

• Evaluating the circuit:
  ‣ Encryption function: SHA-1 of key information XORed with plaintext
  ‣ Diffie-Hellman based OT
What could go wrong?

- What if I terminate the protocol early?
- What if I give the wrong input values to the evaluator?
- What if I garbled the wrong function?
Adversary models

• Semi-Honest model
  ‣ Everyone plays by the rules

• Covert Model
  ‣ Only deviates if malicious behavior is secret

• Malicious Model
  ‣ Deviates arbitrarily
Some tips for building into the malicious model

- Cut-&-Choose
- Zero-Knowledge consistency proofs
- Committing OT
Why don’t we see this in practice?

- Size of circuits
  - On the order of GB

- Hundreds of circuit copies evaluated for malicious security
  - \(2^{-0.32k}\) where \(k = \) number of circuits generated

- OT constructions & malicious model proof systems require very costly group operations
Techniques for speeding things up

- Pipelining
- Parallelization
- Smarter Circuit construction
  - Free XOR
  - Circuit templates
- Outsourcing
How far have we come since Fairplay?

- Millionaire’s problem (Semi-honest Security):
  - 2.4 GHz single core
  - 254 gates
  - 1.25 seconds

- AES (Malicious Security):
  - 256 cores @ 2.3 GHz
  - 49,000+ gates
  - 1.4 seconds