T-79.514 Special Course on Cryptology

Seminar 2: Intro to Cryptography

Helger Lipmaa
Helsinki University of Technology
http://www.tcs.hut.fi/~helger

Overview of This Talk

- Cryptography for data-miners
- Stress on PPDM, generality
- Easy enough (?) for data-miners
- Hopefully not completely boring for cryptographers

Introduction to the Area: Buzzwords

Thanks to www.googlism.com!

- *Datamining* is an automated process for discovering information in large data sets to be used in decision, datamining is alive and well on the internet, datamining is all about counting, datamining is perfectly legal, *datamining is using a database to gain more information about your business*

- *Cryptography* is related with the communication or computation involving two or more parties who may not trust one another, cryptography is the most powerful single tool that users can use to secure the internet, cryptography is outlawed, *cryptography is the art of hiding the meaning of information*

History of Cryptography: Dark Ages

- Cryptography = art of concealing the meaning
- First attempts: invention of the script
  - ⋆ Often, only priests could read
- Use in wars: Sparta, Caesar, middle ages
- In WW2, success of Allies in cryptanalysis is often said to be a decisive factor in “quick” win

Next Step: Public Cryptography

- In early 70s, the importance of cryptanalysis in WW2 was revealed
- At the same time, a call for the first open competition for any kind of cryptographic primitive was published
  - ⋆ In early 1977, IBM's DES was chosen as the US governmental block cipher standard for nonclassified tasks
- 1976: Diffie and Hellman published a seminal paper on public-key cryptography
  - ⋆ 1997: PKC was invented about 5 years earlier in British secret service but never published

Modern Cryptography: Seventies, Eighties

- 1979: Secret Sharing, 1979–1981: Chaum started to work on mix-nets, e-cash, e-voting, that is, in *protocols*

- Eighties: Work on foundations. Defination approach: (a) define what do you want, (b) prove that this can be achieved in theory ("probabilistic polynomial-time"), (c) prove that nothing better can be achieved (i.e., that you cannot have cryptographic primitives that satisfy stricter security definitions).

- Notable achievements: understanding and defining of basic security notions, zero-knowledge (one of the most amazing results in theoretical computer science, and may be also in mathematics, during the last 25 years), multi-party computation.

Modern Cryptography: End of eighties

- Cryptographers had a firm understanding of what is possible in theory.
- Published example protocols were usually proofs of concepts, not meant to be applied.
- Cryptography was firmly based on reductions:
  - ⋆ Prove that if A is secure then B is secure; or if B can be broken then A can also be broken.
- Makes it possible to construct complicated protocols, assuming only that (say) one-way or trapdoor functions exist, factoring or discrete logarithm is hard.

Postmodern Cryptography: Nineties, 2000+

- Exact reductions: If B can be broken in time t with probability ε then A can be broken in time, close to t, with probability, close to ε.
- Efficiency: minimize the resources, needed to execute the protocols.
- Holy grail: construct efficient protocols that have exact reductions to minimal primitives.

Hypermodern Cryptography: Now

- Efficient protocols for many real-life problems are known.
- For other problems, it can be sometimes shown that no efficient solutions exist.
- Fundamental problems, again: a lot of cryptography would collapse if P=NP, or even if P≠NP but one-way functions do not exist. Many protocols would collapse if one could factor efficiently.
- Thus, cryptography has solid foundations *under the assumptions like factoring is hard*.

Do Cryptographers Dream of Quantum Computers?

- 1994: Shor showed that factoring and discrete logarithm can be solved efficiently on a quantum computer
- Fortunately (?), it is not known whether one can actually build a quantum computer. (Do the laws of physics allow it?)
- But even so, it is fundamentally difficult to prove anything about hardness of algorithms!

Resource Bounded Unprovability of Computational Lower Bounds

Tatsuaki Okamoto and Ryo Kashima

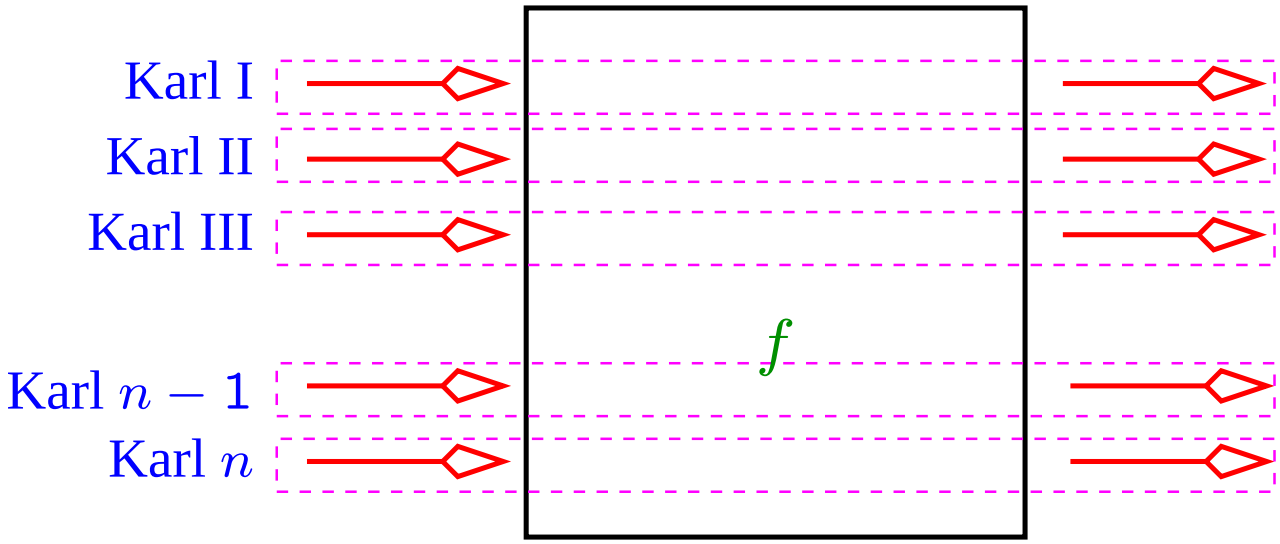
Abstract. This paper shows that no polynomial-time Turing machine can produce a proof (based on a reasonable theory including Peano Arithmetic) of a super-polynomial-time lower bound of an NP (or more generally, PSPACE) problem. In other words, no polynomial-time Turing machine can produce a proof of "P $\neq$ NP". Therefore, *to prove "P $\neq$ NP"* (by any technique and any reasonable theory) *requires super-polynomial-time computational power*. This result is a kind of generalization of the result of 'Natural Proofs" by Razborov and Rudich, who showed that to prove "P $\neq$ NP" by a class of techniques called "Natural" implies computational power that can break a typical cryptographic primitive, a pseudo-random generator. This result also implies that *there is no (finite-size) formal proof for "P $\neq$ NP"* in any reasonable theory. This is considered to be a generalization of the result by Baker, Gill and Solovay, who showed that there is no relativizable proof for "P $\neq$ NP". Based on this result, we show that *the security of any computational cryptographic scheme is unprovable* in the standard setting of modern cryptography, where an adversary is modeled as a polynomial-time Turing machine.

eprint archive, http://eprint.iacr.org/2003/187/, received 9 Sep 2003

Back to Multi-Party Computation

- Main result for us: all efficiently computable functions can also computed securely
- Assume there are n participants, and the ith participant has input x_i. Assume f is a function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$.
- There is a way (*multi-party computation*) to compute f so that at the end of the protocol, the ith participant will get the know value of y_i and nothing else, except what she could compute from (x_i, y_i) herself.

We Gotta Have Some Pictures



Assume f is any function. Karl's can compute f so that (a) Security: Karl i obtains the output he wanted to obtain, (b) Privacy: Karl i will not obtain any new information that cannot be computed from his input and output alone.

Applications: Millionaire's Problem

- Alice and Bob want to know, who is richer, without revealing their private inputs.
- Denote their inputs as x_A and x_B.
- Security: Alice and Bob get to know the value of the predicate $y_A, y_B := [\mathsf{cmp}(x_A, x_B)]$.
- Privacy: Alice will not get any new information that she cannot compute from x_A and y_A. Ditto for Bob.

Applications: Voting

- n voters, one tallier.
- Voter i has input v_i, her vote.
- Security: Tallier gets to know $y_T := \sum_{i=1}^{n} v_i$.
- Privacy: Tallier will not get any information that cannot be computed from y_T alone. Voters will not get any new information at all.

Applications: Data-mining

- Assume you have some data-mining algorithm $\mathcal{A}$, that based on a database $\mu = (\mu_1, \ldots, \mu_n)$, says something interesting about it, $\mathcal{A}(\mu)$.

- Many different different settings, two of them are:

  1. Alice is a client who makes a query, Bob owns the whole database.

  2. Parts of database ("vertical" or "horizontal") sharing are owned by different parties, who want to discover something about the joint of their databases.

- All settings have their natural security and privacy definitions.

Limitations

- MPC: To get total privacy and security, a majority of the parties must be honest (in some settings, $2/3$!)
- Two-party computation: privacy possible, but security is possible only for one of the two parties (since he can halt as soon as he recovers his output)
- Fortunately, often one can design protocols, where halting is not a problem — but not always
- Must assume certain unproven hypotheses (existence of one-way functions, . . .)

Last example: Vickrey Auctions

- Idea: highest bidder pays the second highest bid
- Good: Pareto-efficient, sealed-bid, incentive-compatible, ...
- Still not used widely in practice
- One of the main reasons for this: insecurity
  - ⋆ auctioneers can change the winner and the winning price undetectably
- High motivation for cryptographic Vickrey auctions

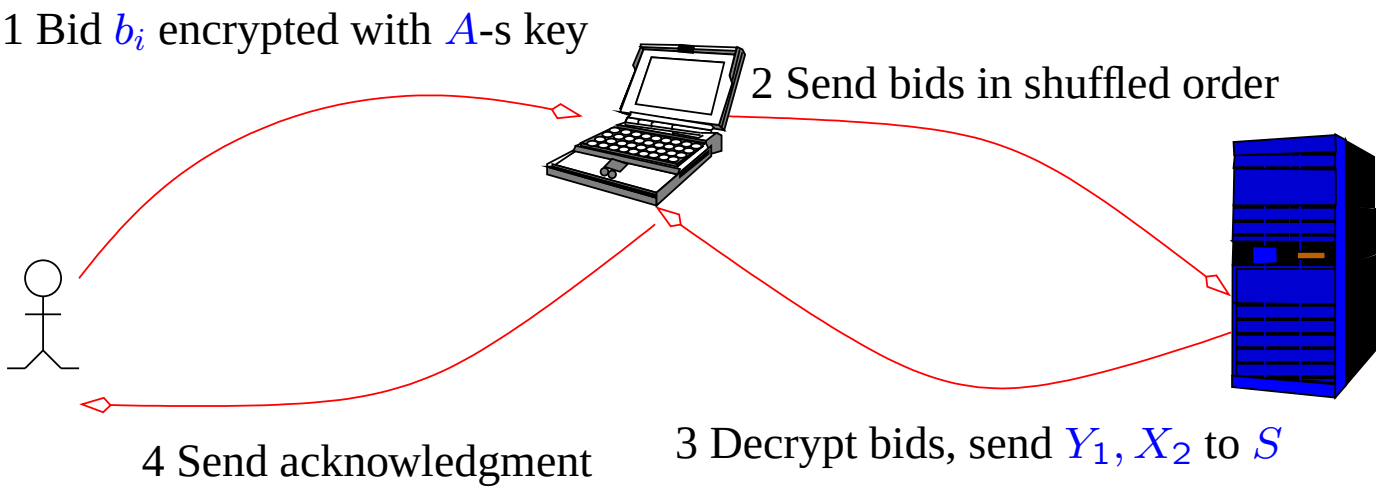
Security model

- Common auctions over Internet have often
  - ⋆ an occassional, *untrusted*, seller with potentially *large number* of bidders
  - ⋆ this seller has a single server, or has supreme control over several servers
- In both cases, *threshold trust ("majority of servers is honest") is not an option*
- Instead, introduce a semitrusted third party, *auction authority* A

Security requirements

- Correctness
  - ⋆ Highest bidder Y_1 should win
  - ⋆ He should pay the second highest bid X_2
- Privacy: S should not get any information about the bids but (Y_1, X_2)
- Scheme should be secure unless both A and S are malicious

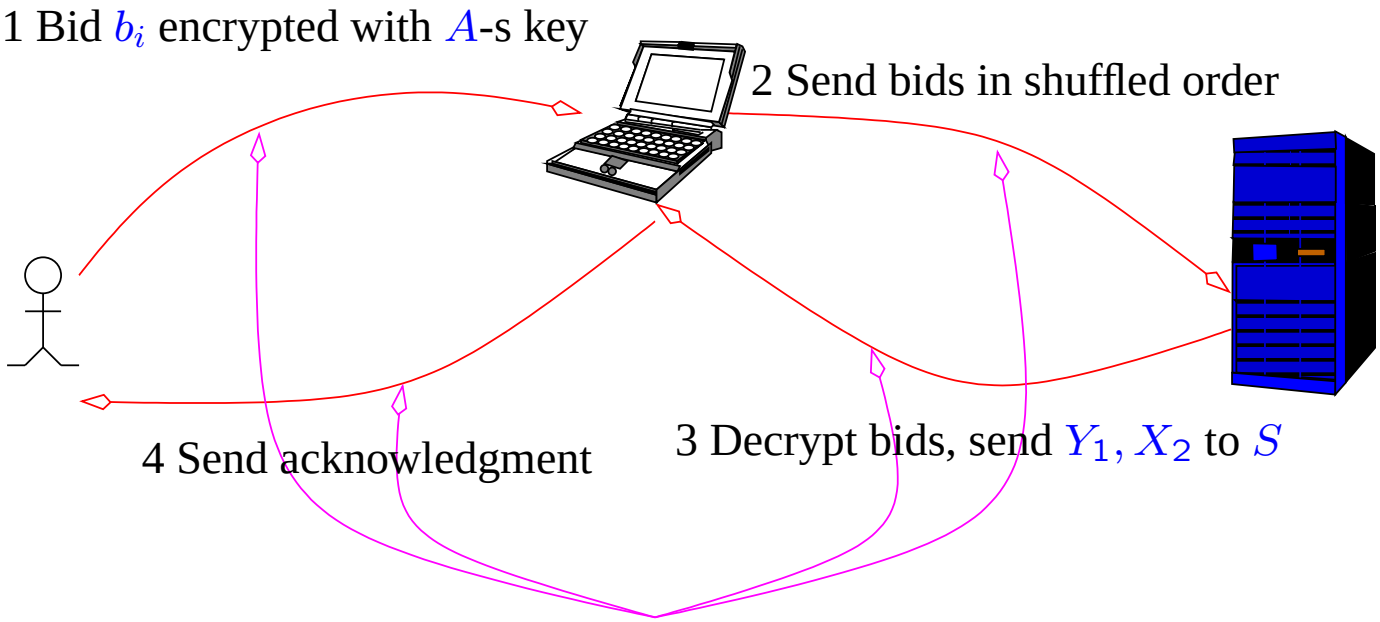
Simple scheme



S will not get any extra information, but S can increase X_2

$A \rightarrow S$ interaction is quite large

Simple scheme $\longrightarrow$ complex scheme



Add correctness proofs

Proofs of correctness

1. Complex: use bulletin board, prove that bid belongs to some set
2. Complex: combine bids, prove correctness of combination
3. Complex: extract X_2, prove it
4. Simple: (Y_1, X_2) signed by S

For more, see my FC 2002 paper with Asokan and Niemi.

Last Remarks

- Cryptographic protocols must be data-dependent: otherwise one can't get privacy
- They are mostly used to add another layer of security ("X is really in the database!) and privacy to the existing algorithms
- In principle, this can be done with all efficient algorithms, but remember the theoretical limitations!
- Designing an efficient protocol for a concrete problem at hand is often white magic.

Questions?

?