Asymptotic algebra and modern cryptanalysis

Alexei Myasnikov

McGill University (Montreal)

Algebraic Cryptography Center (Stevens Inst.)

1

In this series of three lectures I am going to discuss two recent developments in modern cryptanalysis: **generic complexity** and **asymptotic dominance**.

The first one concerns with behavior of algorithms on most typical or "generic" inputs, while the second one deals with the asymptotically most dominant properties of algebraic objects.

The goal is three-fold:

to develop a framework (standards?) for rigorous cryptanalysis of algebraic cryptosystems;

to study new interesting pure algebraic problems coming out of modern cryptography;

to study generic complexity.

Plan

Lecture 1. Generic complexity and cryptanalysis

Lecture 2. Generic complexity of the classical algorithmic problems in groups

Lecture 3. Asymptotic dominance and cryptanalysis

Generic complexity and cryptanalysis

Plan

1. Schemes of public key exchange
2. Generic complexity
3. Generic vs standard complexity measures
4. Black Holes

1. Schemes of public key exchange

- Quest for new cryptosystems
- Group based cryptosystems
- Security assumptions

Public key exchange: general description

Alice and Bob want to exchange messages publicly (via the internet) to get a private (secret) shared key:

Alice chooses a secret key a, encodes it into a^* and sends a^* publicly to Bob.

Bob chooses a secret key b, encodes it into b^* and sends b^* publicly to Alice.

Alice and Bob compute (separately) a shared private key p.

Remark: the functions of encoding and computing the shared key are public.

Diffie-Hellman Key Exchange

Public: a prime number p and a number q.

Alice: chooses a private number (key) $a \in \mathbb{N}$ encodes and sends:

$$a^* = q^a (mod\ p)$$

Bob: chooses a private number (key) $b \in \mathbb{N}$ encodes and sends:

$$b^* = q^b (mod\ p)$$

Shared secrete key:

$$(b^*)^a = (q^b)^a = q^{ba} = (q^a)^b = (a^*)^b = q^{ab} (mod\ p)$$

Why Practical?

"Easy" to compute $a^* = q^a (mod\ p)$.

Why secure?

Hard to solve equations of the type

$q^x = a^* (mod\ p)$,

i.e., the **discrete logarithm** function

$$x = log_q a^*$$

is **"hard"** to compute in $\mathbb{Z}_p$.

RSA public key exchange:

RSA = Rivest-Shamir-Adleman (1978)

Public: n, e

Encoding: $m \to m^* = m^e (mod\ n)$

Security: hard to solve equations of the type

$$x^e = m^* (mod\ n)$$

i.e., the **discrete root** function

$$x = (m^*)^{1/e} (mod\ n)$$

is hard to compute.

Usually,

$$n = pq$$

where p, q are primes.

If p, q are known then computing

$$x = (m^*)^{1/e} (mod\ pq)$$

is much easier.

The *discrete root* function is easy modulo the

Factorization Problem: *decompose n into a product of primes.*

Quest for new cryptosystems

Recent concerns with RSA:

Encoding is not fast enough - secret keys are too long;

Security of RSA is not known

Factorization Problem looks weaker than before:

primality test is in polynomial time;

fast quantum algorithms for the Factorization Problem;

if RSA is broken tomorrow what will we do?

Group based cryptosystems

To be concrete I will discuss one particular non-commutative cryptosystem (based on groups), though the methods apply to the general case.

Base of the cryptosystem:

A finitely generated group $G = \langle X \rangle$

Hiding Machine: a given algorithm to compute "normal forms" of elements of G (viewed as words in generators $X \cup X^{-1}$)

$$w \in (X \cup X^{-1})^* \to \bar{w} \in (X \cup X^{-1})^*$$

Group-based Diffie-Hellman (Ko-Lee, 2000)

Public: A group G, a fixed element $q \in G$,

and two finitely generated subgroups $A, B \subset G$ with $[A, B] = 1$.

Alice takes (secret) $a \in A$, encodes $a \to \overline{q^a} = a^*$, and makes a^* public.

Bob takes (secret) $b \in B$, encodes $b \to \overline{q^b} = b^*$, and makes b^* public.

Common (private) key:

$$(b^*)^a = (q^b)^a = q^{ba} = q^{ab} = (q^a)^b = (a^*)^b$$

Anshel-Anshel-Goldfeld (1999)

New idea (does not exist in commutative case).

Public: Group G with two finitely generated subgroups

$$A = \langle a_1, \ldots, a_m \rangle, \quad B = \langle b_1, \ldots, b_n \rangle$$

Alice: takes a (secret) element $a \in A$ as a word $a = u(a_1, \ldots, a_m)$ in the generators of A, computes the conjugates

$$b_1^a, \ldots, b_n^a$$

encodes them (by the normal forms), and sends publicly.

Bob: takes a (secret) element $b \in B$ as a word $b = v(b_1, \ldots, b_n)$ in the generators of B, computes the conjugates

$$a_1^b, \ldots, a_m^b,$$

encodes them (by normal forms), and sends publicly:

To decode:

Alice computes $a^b = u(a_1^b, \ldots, a_m^b)$

Bob computes $b^a = v(b_1^a, \ldots, b_n^a)$

Shared **secret key**:

$[\mathbf{a}, \mathbf{b}] = a^{-1}(b^{-1}ab) = \mathbf{a^{-1}a^b}$

$[\mathbf{a}, \mathbf{b}] = (a^{-1}b^{-1}a)b = \mathbf{(b^a)^{-1}b}$

Requirements on the Hiding Machine

The set of normal forms in G has to satisfy the following conditions:

- computing $w \longrightarrow \bar{w}$ is "easy".
- "hard" to recover w from $\bar{w}$.

Security of AAG scheme

The Conjugacy Problem (CP) in G is "hard" ...

More precisely, the **Search Conjugacy Problem (SCP)**, i.e., the problem of finding a solution of a conjugacy equation $w^x = w^*$, is "hard" in G.

Notice, (SCP) may be much harder then (CP)!

Even more precisely, the **Search Multiple Conjugacy Problem (SMCP)**

$w_1^x = w_1^*, \ldots, w_k^x = w_k^*$

is "hard"

Security of AAG scheme

Even more precisely:

The Search Multiple Conjugacy Problem

$w^x = w^*$

with constraints $\mathbf{x} \in \mathbf{A}$ has to be "hard" in G.

Homework: check that solution of the SMCP with constrains breaks the system.

"Hardness" of algorithmic problems

Ideally for cryptography - hard to break for each key:

"hard" = "hard on all inputs"

Impossible! One can always store some answers in the memory.

Still good:

"hard" = "hard on most inputs"

Outcome of this discussion

Cryptography quest: find a generically hard problem.

One needs **Generic Complexity** of algorithmic problems.

Generic complexity = complexity on most inputs.

Generic sets

Let S be a set with a fixed measure μ.

A subset $Q \subset S$ is **generic** if $\mu(Q) = 1$,

Q is **negligible** if $\mu(Q) = 0$.

Note: The choice of μ is very important, it should be natural in the context of the problem (no cheating, no funny measures).

Often S occurs via some generating procedure $\mathcal{G}$ (random key generator), in this case μ is the resulting distribution on S.

Generic complexity of algorithms

Let P be an algorithmic problem with the set of input S.

A partial algorithm $\mathcal{A}$ (with inputs from S) **generically solves** the problem P if the halting set $S_{\mathcal{A}}$ (where $\mathcal{A}$ halts) is generic in S and $\mathcal{A}$ solves P on $S_{\mathcal{A}}$.

A time function $f(n)$ is a **generic upper bound** for $\mathcal{A}$ if there exists a generic set $Q \subset S_{\mathcal{A}}$ such that for each $w \in Q$

$$T_{\mathcal{A}}(w) \leq f(|w|)$$

where $|w|$ is the *size* of w, and $T_{\mathcal{A}}(w)$ - the number of steps required for $\mathcal{A}$ to stop on w.

Asymptotic density

A **stratification** of S is a representation

$$S = \bigcup_{i=0}^{\infty} S_i$$

of S as union of an increasing chain of subsets of S.

The limit (if it exists)

$$\rho(Q) = \lim_{i \to \infty} \frac{\mu(Q \cap S_i)}{\mu(S_i)}$$

is called the **asymptotic density** of a subset Q of S (with respect to the chain $\{S_i\}$ and the measure μ).

Q is **generic** (with respect to $\{S_i\}$ and μ) if $\rho(Q) = 1$.

Asymptotic density and size of inputs

Usually stratifications

$$S = \bigcup_{i=0}^{\infty} S_i$$

correspond to a fixed **size function** $size : S \to \mathbb{R}$ on S, so

$$S_i = \{w \in S \mid size(w) \leq i\}$$

In this case

$$\frac{\mu(Q \cap S_i)}{\mu(S_i)}$$

is the **probability** of an element of size i from S to be in Q.

Size functions

Again, the size functions on S have to be natural in the context of the problem in hand.

In general size of an input $w \in S$ is the time required for the generating procedure $\mathcal{G}$.

Example: Whitehead problem in free groups.

Various types of generic sets

Generic sets may have different "sizes" : some are bigger then the others. Classification of generic sets by size is given in "Multiplicative measures on groups" (**Borovik, Myasnikov, Remeslennikov**).

Here I mention only one, very crude distinction:

recall, that $Q \subseteq S$ is generic if

$$\frac{\mu(Q \cap S_i)}{\mu(S_i)} \to 1$$

Now, Q is *strongly generic* if

$$\frac{\mu(Q \cap S_i)}{\mu(S_i)} \to 1$$

exponentially fast when $i \to \infty$.

Example:

subgroups of infinite index are generic in a free group F;

normal subgroups with non-amenable factor-groups are strongly generic.

Wild generic sets

Theorem [Myasnikov, Ushakov] Let F be a non-abelian free group. Then there is a subset $S \subset F$ and an automorphism $\alpha \in Aut(F)$ such that S is generic and S^{α} is negligible.

Hence, genericity of a set $S \subset F$ may depend on the choice of the basis in F.

Generic vs standard complexity measures

Generic complexity was introduced by Kapovich, Myasnikov, Schupp, and Shpilrain in two papers were we studied generic and average complexities of the classical algorithmic problems in groups.

I will mention some of these and other results in due course.

Robert Gilman in his talk later today will discuss generic complexity of several classical problems from complexity theory.

Alexander Rybalov will discuss some new results on generic complexity of the Halting Problem for Turing machines in his talk tomorrow.

NP-complete problems

Karp, Cook

$\mathbf{P}$ = polynomial time

"Polynomial time = tractable"

$\mathbf{NP}$ = polynomial time by non-deterministic Turing machine.

Many real problems are in $\mathbf{NP}$.

$\mathbf{NP}$-complete problems = the hardest in $\mathbf{NP}$

3-SAT, TSP, Hamilton cycle,...

Common belief: NP-complete problems are hard!

Generic vs NP-complete

Claim: Typical NP-complete problems are **generically easy!**

3-SAT, Hamilton cycle problem, ... - much more details in Gilman's talk.

Reason: Classical time complexity is complexity on the *worst inputs*, which are typically *very very sparse*.

Classical example: the Dantzig's Simplex Algorithm for linear programming is of exponential time (worst case) but nevertheless it is used hundreds of times daily and in practice almost always works quickly.

NP-completeness is not a proper measure for crypto security

Average case complexity

Let μ be a measure on the set of inputs S of the algorithm $\mathcal{A}$.

Expected running time of the algorithm $\mathcal{A}$ on the set S:

$$\int_S T_{\mathcal{A}}(w)\mu(w)$$

Levin (1986), Gurevich (1987)

Average P and NP.

Average case NP-complete problems.

Common belief: Average case NP-complete problems are very hard!

Average Case vs Worst Case:

Gurevich - Shelah (1987):

To find a Hamilton cycle (a closed path that contains every vertex exactly once) in a finite graph is linear time on average.

Some NP-complete problems are linear on average!

Generic vs Average:

Average Case: the algorithm solves the problem on **all inputs**

Generic Case: the algorithm solves the problem only on a **generic subset**

There are **exponential on average** problems with **linear** generic case complexity.

Theorem [Gurevich-Shelah, Hamkins-Myasnikov] There are **NP-complete on average** problems with **polynomial generic case** complexity.