SAC Summer School 2020

Polynomial Equations in Cryptology

Lecture by Antoine Joux

October 20th, 2020

Part 1

Motivation and Background

Writing cryptographic problems as systems of equations

- Generically (as problems in NP) by converting 3-SAT to Boolean systems
- Or directly from the description of the problem. Examples:
  - RSA with exponent 3: Solve $X^3 - m = 0 \pmod{N}$
  - DES, writing the circuit as Boolean polynomials in the *Key* variables
  - Path of 2-isogenies from j_0 to j_ℓ:

$$\Phi_2(j_0, j_1) = 0,\ \Phi_2(j_1, j_2) = 0,\ \ldots,\ \Phi_2(j_{\ell-1}, j_\ell) = 0$$

  - Discrete log:

$$(1 + x_0(g - 1)) \cdot (1 + x_1(g^2 - 1)) \cdots (1 + x_k(g^{2^k} - 1)) - h = 0 \pmod{G}$$

with $x_i^2 - x_i = 0$

Breaking some cryptographic problems as systems of equations

- If the corresponding system is easy to solve, the system is broken

  - RSA with exponent 3 (many recipients):
    Solve $X^3 - m = 0$ over $\mathbf{Z}$

  - Unfiltered linear shift back shift registers
    Linear algebra over GF(2)

  - Discrete log. and Isogenies between Drinfeld modules
    Linear algebra

Systems of polynomial equations

- Many cases, plenty of parameters:
  - Over finite fields, over the reals/complex, over Rings
  - Linear vs. non-linear systems of equations
  - Number of variables
  - Sparse or dense systems
  - Arbitrary or structured solutions (such as « small » solutions)
  - Over or under-determined systems
  - One solution vs. all solutions

- We need to distinguish the easy from the hard cases
  - Main goal of this lecture

CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

A review of easy cases

Linear systems

- Easiest and best known case
  - Linear systems A.x=b are easy to solve (at most cubic in dimension)
  - Over any field. (Over rings, some care is required)
  - Either one solution or a compact description of all solutions

- Sparse linear systems can also be considered
  - Gaussian elimination becomes problematic
  - Instead, structured elimination and iterative algorithms are used

Univariate polynomials over finite fields

- Finding roots is efficient (in the degree of f)

- Remember that $\forall \alpha \in \mathbb{F}_p : \alpha^p = \alpha$
  - Thus, any root of f in $\mathbb{F}_p$ is also a root of $X^p - X$ => use (efficient) Gcd

  - Also: $X^p - X = X \cdot (X^{(p+1)/2} + 1) \cdot (X^{(p+1)/2} - 1)$ => Gcd again

  - Complexity of most efficient methods follows $\deg(f)^{3/2}$

Univariate polynomials over rings $\mathbf{Z}_N$

- Basic idea is to

$$\text{write: } N = \prod_{i=1}^{r} p_i^{e_i}$$

- If N is square-free, just apply CRT

- Otherwise, Hensel lifting to get roots modulo $p_1^2, p_1^3, \ldots$

- However, requires a factoring algorithm !

Polynomials over $\mathbf{Z}$

- For the univariate case, find reals roots (and check if near an integer)

- With more variables, we have a Diophantine equation
  - In general, they are extremely hard to solve

- For the bivariate case, it remains difficult in general
  - Even special case of Thue's equations is already computationally intensive

- By contrast, over $\mathbf{F}_p$ finding one solution of low degree polynomial equation in many variables, is usually easy (Guess and solve)

Coppersmith small roots algorithms

- Coppersmith’s first algorithm find small solutions of a low-degree bivariate Diophantine equation
- Coppersmith’s second algorithm find small solutions of a low degree univariate polynomial modulo N (without factoring)
- Based on lattice reduction (on some kind of Macaulay’s matrix)

End of Part 1

CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

Part 2

Hard cases and algorithms

Linear systems (with noise)

- Adding noise turns an easy problem into a seemingly hard one

- Can be transformed into a system of polynomial equation of higher degree
- Indeed, we have

$$AX + E = B \text{ with known } A, B \text{ and small } E$$

- Write the smallness condition as polynomial equations
- For example, if error are -1, 0 or 1 then

$$\forall i : \; x_i^3 - x_i = 0$$

- Larger errors require higher degree

(Sparse) Univariate polynomials over finite fields

- Sparsity seems to make things difficult

- Can be transformed into a system of polynomial equations of higher degree

- For example, take $f(X) = X^{1025} + X^{69} + 1$ and write:

  $X_1 - X^2 = 0, X_2 - X_1^2 = 0, \cdots, X_{10} - X_9^2 = 0$ and

  $X_{10} \cdot X + X_6 \cdot X_2 \cdot X + 1 = 0$

Multivariate systems of polynomial equations

- This is a NP-hard problem
- Tons of applications:
  - Algebraic geometry, robotics, …

- In crypto, we are mostly interested by equations over finite fields
  - Don’t need to worry about precision or coefficients explosion

- The case of GF(2) is particularly interesting

Polynomial systems: the Boolean case

Solving systems over GF(2) a.k.a. Boolean systems

$$
\begin{aligned}
f_1(X_1, \ldots, X_n) &= 0 \\
f_2(X_1, \ldots, X_n) &= 0 \\
&\vdots \\
f_m(X_1, \ldots, X_n) &= 0
\end{aligned}
$$

Initial system: degree (at most) d

$$
\begin{aligned}
X_1^2 + X_1 &= 0 \\
X_2^2 + X_2 &= 0 \\
&\vdots \\
X_n^2 + X_n &= 0
\end{aligned}
$$

Implicit field equations (degree 2)

Three parameters: n, m and d

Another application: Multivariate cryptography

- The hardness of boolean systems has been used as a building block

- **Main idea:** boolean equation systems with a trapdoor
  - F(x)=y hard to solve for the public representation of F
  - F(x)=y easy to solve with the private representation

  - **Signature:** solve F(signature)=Hash(random, message)
  - **Encryption:** recover unique secret with F(secret)=ciphertext

- Solving Boolean systems can break such systems
- Recovering the trapdoor offers another direction of attack

Example of Multivariate cryptography HFE (Hidden Field Equation)

Private equation

$$F(X) = Y \text{ in } \mathbb{F}_{2^n}$$

Trapdoor

Take α s.t. $\mathbb{F}_{2^n} = \mathbb{F}_2[\alpha]$

Write $X = \sum_{i=0}^{n-1} x_i \alpha^i$

Express F as coordinates and mix

Public system

A priori, random looking

n eqs in n vars

Degree depends on F

Exhaustive Search

Exhaustive search

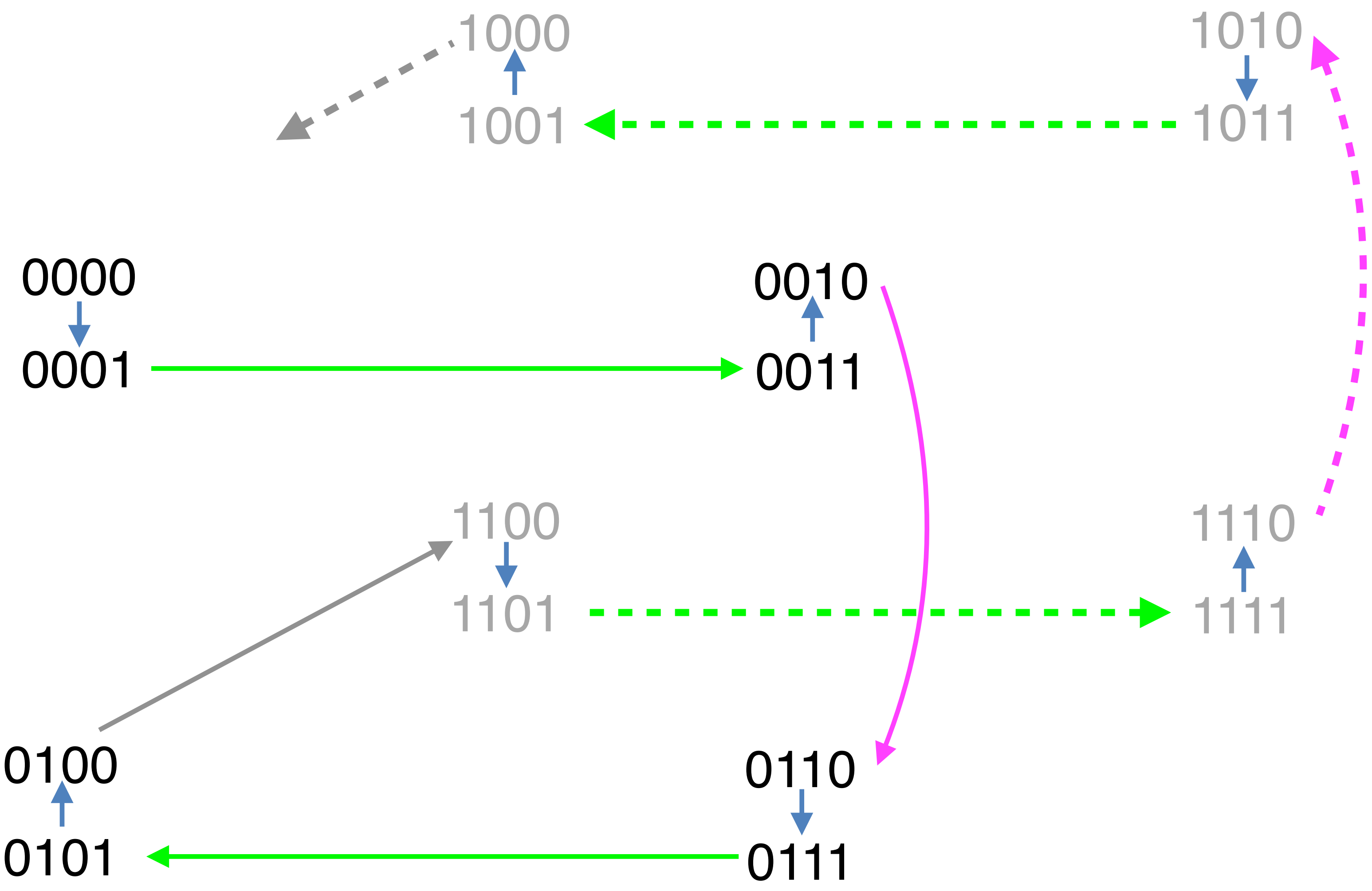
- A priori: 2^n evaluations of the m polynomial functions

- Many optimisations
  - Evaluate f_{i+1} only for assignments OK on $f_1, f_2, \cdots, f_i$
    - Expected cost bounded by $2^n + 2^{n-1} + \cdots + 1 + (m-n) < m\,2^n$
  - Evaluate in parallel (bitslicing) on length of CPU registers

  - Use Grey codes and modify only one variable between two evaluations
  - Use derivative to do the modification faster

  - State-of-the-art software implementation in LibFes by Bouillaguet

Grey codes (example on 4 bits)



Fast updating of evaluations

- When changing x_i write $F(x_1, \cdots, x_n)$ as

$$F_0^{(i)}(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n) + x_i \cdot F_1^{(i)}(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)$$

- To update x_i we just need to add:

$$F_1^{(i)}(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)$$

- Note that this is a polynomial of lower degree
- Repeat the trick to also update those

- Note that $F_1^{(i)}$ is a partial derivative of F

Probabilistic polynomials

Evaluation of probabilistic polynomials

- Introduced by Lokshtanov–Paturi–Tamaki–Williams–Yu (SODA 2017)
- Only looks at n (does not gain from overdetermination)

- Consider a random polynomial:

$$R_r = 1 + \sum_{i=1}^{m} r_i f_i(x_1, \cdots x_n)$$

- Its value is 1 at any solution of the system and a random 0/1 elsewhere

Evaluation of probabilistic polynomials (2)

- Take several random polynomials $R_1, \cdots, R_\ell$ of the form:

$$R_r = 1 + \sum_{i=1}^{m} r_i f_i(x_1, \cdots x_n)$$

- The product

$$P = \prod_{i=1}^{\ell} R_i$$

has value 1 at every solution and value 0 w.h.p at every non-solution

- Here, w.h.p. (with high probability) is $1 - 2^{-\ell}$

Evaluation of probabilistic polynomials (3)

- Now, split the variables in two groups: $x_1, \cdots, x_t$ and $x_{t+1}, \cdots, x_n$
- Now compute

$$Q = \sum_{a_1=0}^{1} \sum_{a_2=0}^{1} \cdots \sum_{a_t=0}^{1} P(a_1, \cdots, a_t, x_{t+1}, \cdots, x_n)$$

- $Q(\alpha_{t+1}, \cdots, \alpha_n)$ counts the parity of the number of n-uples ending with $\alpha_{t+1}, \cdots, \alpha_n$ where P takes value 1 (mostly solutions if $t > \ell$)

- We want to use this to find solution. Minor problem=multiple sols in block
  - Not a problem if unicity guaranteed
  - Otherwise, randomise Q again or add equations to force unicity [follow-up by Björklund-Kaski-Williams]

Evaluation of probabilistic polynomials (4)

- We now have a methods that computes

$$Q = \sum_{a_1=0}^{1} \sum_{a_2=0}^{1} \cdots \sum_{a_t=0}^{1} P(a_1, \cdots, a_t, x_{t+1}, \cdots, x_n)$$

  then finds solutions in $x_{t+1}, \cdots, x_n$ of $Q = 1$

- It works by computing Q formally then doing fast evaluation at all points
- The exhaustive search part is reduced to 2^{n-t}
- Q-preparation cost is the number of monomials of degree ℓd in $x_{t+1}, \cdots, x_n$

- With follow-up improvement, the full cost is $O(2^{0.804n})$

Linearisation and Gröbner bases techniques

Overdetermined systems and linearisation

- Degree d=2, assume m>n(n+1)/2

- View each monomial $x_i x_j$ as an independent variable
- We get a linear system of m equations in n(n+1)/2 variables

- Easy to solve
  - And check the non-linear constraints $x_i x_j = x_i \times x_j$

- Can a similar idea be used for m close to n ?

XL, Gröbner bases and Macaulay matrices

- A extension of linearisation was proposed by Kipnis, Shamir (1999)

- In a nutshell, multiply every equation by many monomials
- Use linearisation to solve this extended system

- It can seen as a specialised rediscovery of Lazard's method (1983)
  - Computes Gröbner basis by doing linear algebra in Macaulay's matrix
  - In the Boolean case, adding field equations implicitly helps !

- A big difficulty is to predict the size of the matrix
  - Heuristic estimates give the expected complexity of the approach

Some algebraic background

- The ideal generated by $f_1, f_2, \cdots, f_m$ in $\mathbb{K}[X_1, \cdots, X_n]$ is the set:

$$I(f_1, f_2, \cdots, f_m) = \left\{ \sum_{i=1}^{n} \mu_i f_i \middle| (\mu_1, \cdots, \mu_m) \in \mathbb{K}[X_1, \cdots, X_n]^m \right\}$$

- Every solution of the system is a zero of every polynomial in $I(f_1, f_2, \cdots, f_m)$

- For every degree D we also define:

$$I_D(f_1, f_2, \cdots, f_m) = \left\{ \sum_{i=1}^{n} \mu_i f_i \middle| (\mu_1, \cdots, \mu_m) \text{ s.t. } \forall i : \deg \mu_i f_i \leq D \right\}$$

Algebraic background continued

- We see that $I_D(f_1, f_2, \cdots, f_m)$ is a subset of $I(f_1, f_2, \cdots, f_m)$
- However, in general, I_D doesn't contain all degree D polynomials of I
- When it happens (for all D) then $(f_1, f_2, \cdots, f_m)$ is called a Gröbner basis of I

- Furthermore, if $I_D(f_1, f_2, \cdots, f_m)$ contains all degree D polynomials, then $I_D(f_1, f_2, \cdots, f_m)$ is a Gröbner basis
- Note that $I_D(f_1, f_2, \cdots, f_m)$ is a vector space, so any (linear) basis works
    - A smaller set called reduced Gröbner basis is usually used

- Furthermore, there always is a (possibly large) D that works

Macaulay's matrix

- Multiplying the initial polynomials by monomials gives a basis:

$$I_D(f_1, f_2, \cdots, f_m) = \left\langle \mu_i f_i \,\middle|\, \mu_i \text{ monomial s.t. } \deg \mu_i f_i \leq D \right\rangle$$

- Macaulay's matrix is the matrix representation in the canonical basis
  i.e: its columns are indexed by monomials
  its rows by labels $\mu_i f_i$

- Linear algebra on the matrix with large enough D gives what we want:
  - Row echelon form => reduced Gröbner basis
  - Solution of the linear system + checks => solution of original problem

- Note that reducing modulo field equations reduces the number of monomials

Macaulay's matrix example

	xyz	xy	xz	yz	x	y	z	1
f_1		1				1	1	1
f_2			1		1		1	1
f_3				1	1			1
xf_1			1		1			
yf_1		1		1				
zf_1	1			1				
xf_2								
yf_2	1	1		1		1		
zf_2								
xf_3	1							
yf_3		1		1		1		
zf_3			1	1			1	

$$\begin{cases} f_1 = xy + y + z + 1 \\ f_2 = xz + x + z + 1 \\ f_3 = yz + x + 1 \end{cases}$$

Empty line since $f_2 = (x+1)(z+1)$

Macaulay’s matrix example (continued)

	xyz	xy	xz	yz	x	y	z	1
f_1		1				1	1	1
f_2			1		1		1	1
f_3				1	1			1
xf_1			1		1			
yf_1		1		1				
zf_1	1			1				
xf_2								
yf_2	1	1		1		1		
zf_2								
xf_3	1							
yf_3		1		1		1		
zf_3			1	1			1	

	xyz	xy	xz	yz	x	y	z	1
Kernel	0	0	1	0	1	0	1	1

	xyz	xy	xz	yz	x	y	z	1
Echelon form	1	0	0	0	0	0	0	0
	0	1	0	0	0	0	0	0
	0	0	1	0	0	0	1	0
	0	0	0	1	0	0	0	0
	0	0	0	0	1	0	0	1
	0	0	0	0	0	1	0	0
	0	0	0	0	0	0	1	1

Gröbner basis $(x+1, y, z+1)$

Predicting the degree D

- Studied intensively in Bardet's PhD thesis (under heuristic assumptions)

    For degree 2 polynomials and m=n we have $D \approx 0.09n$

    This gives a complexity of $O(2^{0.873n})$ assuming quadratic time lin.alg.

- D decreases for over-determined systems with $m = \alpha n$ it becomes

$$D \approx n\left(-\alpha + \frac{\left(1+\sqrt{2\alpha^2 - 10\alpha - 1 + 2(\alpha+2)\sqrt{\alpha(\alpha+2)}}\right)}{2}\right)$$

Hybrid methods

- BooleanSolve (Bardet, Faugère, Salvy, Spaenlehauer)
  - Enumerate the values of $0.55\alpha n$ variables then run GB
  - With $\alpha < 1.82$ improves complexity to $2^{(1-0.208\alpha)n}$
  - Not better in practice than exhaustive search

- Crossbred approach (J., Vitse)
  - Change linear algebra in Macaulay's matrix
    - Get polynomials which are linear in k variables (higher in the others)
  - Do exhaustive search of the n-k other variables
    - Solve the resulting linear system
    - Beats exhaustive search already for m=n=40

SAT solvers

SAT solvers

- Convert polynomials into clauses via extra variables
- Proposed by Gregory Bard (unpublished see his webpage)
- Implemented in Magma (calling minisat)
- Works especially well for sparse polynomials

CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

End of Lecture