# Cryptanalytic Applications of the Polynomial Method for Solving Multivariate Equation Systems over GF(2)

Itai Dinur

Ben-Gurion University

EUROCRYPT 2021

# Solving Polynomial Systems

- Input: polynomial system $E = (P_1(X),\ldots,P_m(X))$ over $\mathbb{F}$
- Each $P_j \in \mathbb{F}[X_1,\ldots,X_n]$ given by **algebraic normal form** (**ANF**)
- Algebraic **degree** bounded by small constant $d$

- Goal: **find solution** to $E$
  - $x=(x_1,\ldots,x_n)$ such that for each $j$: $P_j(x)=0$

- For $d=1$: solved in poly-time (Gaussian elimination)
- NP-hard for $d=2$ even over $\mathbb{F}_2$

# Prior Work

- Standard technique: find convenient representation of ideal generated by polys (**Gröbner basis**)
- Complexity analysis heuristic

- At SODA 2017 Lokshtanov et al. [LPTWY] presented **first worst-case** algorithms with exponential speedup over brute force

- Based on **"polynomial method"** in circuit complexity
  - Technique for proving lower bounds, recently applied in algorithm design

# Solving Polynomial Systems over $\mathbb{F}_2$

- Input: polynomial system $E = (P_1(X),\ldots,P_m(X))$ over $\mathbb{F}_2$
- Each $P_j \in \mathbb{F}_2[X_1,\ldots,X_n]$ given by **algebraic normal form** (**ANF**)
- Algebraic **degree** bounded by small constant $d$

- Problem **NP-hard** even for $d=2$
- Assuming Exponential Time Hypothesis, **no sub-exponential** algorithm

- Fundamental problem in computer science
- Widely studied in **cryptography**
  - Multivariate cryptosystems

# Prior Work

| Authors | Year | Technique | Complexity d=2 | Complexity d≥2 | Comments |
|---|---|---|---|---|---|
| Bouillaguet et al. | 2010 | Exhaustive search | $4 \log n \cdot 2^n$ | $2d \log n \cdot 2^n$ | Fast in practice |
| Bardet et al. | 2013 | Hybid (Gröbner basis-related) | $O(2^{0.792n})$ heuristic | - | - |
| Joux and Vitse | 2017 | Hybid (Gröbner basis-related) | - | - | Fast in practice |
| [LPTWY] | 2017 | Polynomial method | $O(2^{0.8765n})$ | $O(2^{(1-1/5d)n})$ | - |
| [BKW] | 2019 | Polynomial method | $O(2^{0.804n})$ | $O(2^{(1-2.7d)n})$ | - |
| [D] | 2021 | Polynomial method | $O(2^{0.6943n})$ | $O(2^{(1-2d)n})$ | - |

# Concrete Complexity

- We consider the **concrete (non-asymptotic) complexity** of solving $\mathbb{F}_2$ equations
- Relevant for choosing parameters of concrete cryptosystems

# Prior Work ($\mathbb{F}_2$) – Concrete complexity

| Authors | Year | Technique | Complexity d=2 | Complexity d≥2 | Comments |
|---|---|---|---|---|---|
| Bouillaguet et al. | 2010 | Exhaustive search | $4 \log n \cdot 2^n$ | $2d \log n \cdot 2^n$ | Fast in practice |
| Bardet et al. | 2013 | Hybid (Gröbner basis-related) | $O(2^{0.792n})$ heuristic | - | Beats brute force for $n \geq 200$ |
| Joux and Vitse | 2017 | Hybid (Gröbner basis-related) | - | - | Fast in practice |
| [LPTWY] | 2017 | Polynomial method | $O(2^{0.792n})$ | $O(2^{(1-1/5d)n})$ | Large overhead |
| [BKW] | 2019 | Polynomial method | $O(2^{0.804n})$ | $O(2^{(1-2.7d)n})$ | Large overhead |
| [D] | 2021 | Polynomial method | $O(2^{0.6943n})$ | $O(2^{(1-2d)n})$ | Large overhead |

# Our Results

- **Concretely efficient** polynomial method algorithm for solving $\mathbb{F}_2$ equations
- Complexity for **random** equation systems (bit operations):
  - $n^2 \cdot 2^{0.815n}$ for $d=2$
  - $n^2 \cdot 2^{(1-2.7d)n}$ for $d \geq 2$
- **Obstacle** for fast practical implementation:
  - **Memory**: $n^2 \cdot 2^{0.63n}$ for $d=2$ ($n^2 \cdot 2^{(1-1.35d)n}$ for $d \geq 2$)

# Prior Work ($\mathbb{F}_2$) – Concrete complexity

| Authors | Year | Technique | Complexity d=2 | Complexity d≥2 | Comments |
|---|---|---|---|---|---|
| Bouillaguet et al. | 2010 | Exhaustive search | $4 \log n \cdot 2^n$ | $2d \log n \cdot 2^n$ | Fast in practice |
| Bardet et al. | 2013 | Hybid (Gröbner basis-related) | $O(2^{0.792n})$ heuristic | - | Beats brute force for $n \geq 200$ |
| Joux and Vitse | 2017 | Hybid (Gröbner basis-related) | - | - | Fast in practice |
| [D] | new | Polynomial method | $n^2 \cdot 2^{0.815n}$ | $n^2 \cdot 2^{(1-2.7d)n}$ | No fast implementation |

# New Algorithm – Concrete Instances

| Variables n | Degree d | Complexity (bit operations) | Exhaustive search (Bouillaguet et al.) |
|---|---|---|---|
| 80 | 2 | $2^{77}$ | $2^{84}$ |
| 128 | 2 | $2^{117}$ | $2^{133}$ |
| 128 | 4 | $2^{129}$ | $2^{134}$ |
| 256 | 2 | $2^{223}$ | $2^{261}$ |
| 256 | 4 | $2^{246}$ | $2^{262}$ |

# Our Results

- **Application**: cryptanalysis of **Picnic** signature scheme
  - Alternate 3rd-round candidate in NIST's **post-quantum** standardization project
  - Based on **LowMC** block cipher
- Some instances of **Picnic-3** do not achieve **claimed security level**

| Security Level | Attack Complexity (bit operations) | Memory (bits) |
|---|---|---|
| 128 | $2^{130}$ | $2^{113}$ |
| 196 | $2^{188}$ | $2^{164}$ |
| 255 | $2^{245}$ | $2^{218}$ |

# Plan

- **Background**
- Overview of the algorithm
- Conclusion and open problems

# $\mathbb{F}_2$ Polynomials

- Notation: $X=(X_1,\ldots,X_n)$ for symbolic variables, and $x=(x_1,\ldots,x_n)$ for assignments

- Equation system

$E=($

$P_1(X_1,X_2,X_3,X_4,X_5) = X_1X_2 + X_1X_3 + X_1X_5 + X_2X_3 + X_3X_5 + X_4X_5 + X_1 + X_4 + X_5$

$P_2(X_1,X_2,X_3,X_4,X_5) = X_1X_3 + X_1X_4 + X_1X_5 + X_2X_5 + X_3X_4 + X_2 + X_3 + X_4 + 1$

$P_3(X_1,X_2,X_3,X_4,X_5) = X_1X_2 + X_1X_3 + X_2X_3 + X_2X_4 + X_3X_4 + X_4X_5 + X_2 + X_4 + 1$

$)$

- System with $n=5$, $m=3$ , $d=2$

# Polynomial Method Algorithms for Solving Equations

- Given $E = (P_1(X),\ldots,P_m(X))$, define
  $F(X) = (P_1(X) + 1)\,(P_2(X) + 1)\ldots(P_m(X) + 1)$
- Assignment $x$ solution of $E \Leftrightarrow F(x)=1$
- Call $F$ **identifying polynomial** of $E$
- $\deg(F) \approx d \cdot m$ too high

# Polynomial Method Algorithms for Solving Equations

- Given $E = (P_1(X),\ldots,P_m(X))$, define
  $F(X) = (P_1(X) + 1)(P_2(X) + 1)\ldots(P_m(X) + 1)$

- Consider a **probabilistic polynomial**
  $\tilde{F}(X) = (R_1(X) + 1)(R_2(X) + 1)\ldots(R_\ell(X) + 1)$ for $\ell < m$
  - Each $R_i(X)$ random linear comb. of $P_j$'s
- $\tilde{F}$ is identifying polynomial of $\tilde{E} = (R_1(X),\ldots,R_\ell(X))$

- 1) $\tilde{F}$ approximates $F$: if $F(x)=1 \rightarrow \tilde{F}(x)=1$
  if $F(x)=0 \rightarrow \tilde{F}(x)=0$ w.p $1-2^{-\ell}$
- 2) $\deg(\tilde{F}) = d\cdot\ell < \deg(F)$

# Polynomial Method Algorithms for Solving Equations

- Consider probabilistic polynomial
- $\tilde{F}(X) = (R_1(X) + 1)(R_2(X) + 1)\ldots(R_\ell(X) + 1)$ for $\ell < m$

- For $n_1 < n$ partition variables to $X=(Y_1,\ldots,Y_{n-n1},Z_1,\ldots,Z_{n1})$

- **Basic algorithm**: compute $U(y)=\Sigma_{z\in\{0,1\}^{n1}} \tilde{F}(y,z)$ for **all** $y\in\{0,1\}^{n-n1}$ **efficiently** (faster than $2^n$)

- Main algorithms uses $U(y)$ to compute solutions to $E$
  - $U(y)$ counts number of solutions mod 2 (parity) to $\tilde{E} = (R_1(y,Z),\ldots, R_\ell(y,Z))$

# Polynomial Method Algorithms for Solving Equations

- **Basic algorithm**: compute $U(y)=\Sigma_{z\in\{0,1\}^{n1}} \tilde{F}(y,z)$ for **all** $y\in\{0,1\}^{n-n1}$ **efficiently** (faster than $2^n$)
- Consider $U(Y)= \Sigma_{z\in\{0,1\}^{n1}} \tilde{F}(Y,z)$ as polynomial in $Y$
  - $\deg(U)\leq \deg(\tilde{F}) = d\cdot\ell$ (actually lower)
- **Sketch of algorithm** (use fast polynomial interpolation\evaluation algorithms):
  - 1) Interpolate ANF of $U(Y)$
  - 2) Evaluate $U(y)$ on all $y\in\{0,1\}^{n-n1}$
- Complexity analysis:
  - 1) $\binom{n-n_1}{\deg(U)}\cdot 2^{n_1}$
  - 2) $2^{n-n_1}$
- Select parameters to balance steps

# Polynomial Method Algorithms for Solving Equations

- **Basic algorithm**: compute $U(y)=\Sigma_{z\in\{0,1\}^{n1}} \tilde{F}(y,z)$ for **all** $y\in\{0,1\}^{n-n1}$ **efficiently** (faster than $2^n$)

- Polynomial method algorithms use parity computations to output solutions to E

- One of our contributions:
- **Simplification** and **optimization** of [BKW] and [D]
  - Simpler and more efficient way of computing solutions to E from parity computations

# Plan

- Background
- **Overview of the algorithm**
- Conclusion and open problems

# Basic Idea

- Given $E = (P_1(X),\ldots,P_m(X))$, define:
  $F(X) = (P_1(X) + 1)(P_2(X) + 1)\ldots(P_m(X) + 1)$

- Probabilistic polynomial
  $\tilde{F}(X) = (R_1(X) + 1)(R_2(X) + 1)\ldots(R_\ell(X) + 1)$ for $\ell < m$
  - Each $R_i(X)$ random linear comb. of $P_j$'s
  - Identifying polynomial of $\tilde{E} = (R_1(X),\ldots,R_\ell(X))$

- Previous algorithms defined many independent probabilistic polynomials that approximate F
  - Large **concrete overhead**

- Observation: each solution of E also solves $\tilde{E}$
- Algorithm: iterate over solutions of $\tilde{E}$, check if solve E

# Iterating over (Many) Solutions of $\tilde{E}$

- For $n_1 < n$ partition variables to $X=(Y_1,\ldots,Y_{n-n1},Z_1,\ldots,Z_{n1})$
- Set parameters s.t. for each $y\in\{0,1\}^{n-n1}$, there is (on average) single $z\in\{0,1\}^{n1}$ s.t $(y,z)$ is solution to $\tilde{E} = (R_1(Y,Z),\ldots, R_\ell(Y,Z))$
  - $\ell \approx n_1$
- Divide and conquer algorithm:
  - For each $y\in\{0,1\}^{n-n1}$ : **"fill in"** $z$ so $(y,z)$ solution of $\tilde{E}$ by **parity computations**, and test $(y,z)$ on $E$
- Recall: $U(y)$ counts parity of solutions to $\tilde{E} = (R_1(y,Z),\ldots, R_\ell(y,Z))$
- If $(y',z')$ is **only solution** to $\tilde{E} = (R_1(y',Z),\ldots, R_\ell(y',Z))$, use $n_1$ "tweaked" parity computations to **"fill in"** $n_1$ bits of $z'$

# Analysis (Sketch)

- Divide and conquer algorithm:
  - For each $y \in \{0,1\}^{n-n1}$ : **"fill in"** solution $(y,z)$ of $\tilde{E}$ by parity computations, and test it on $E$

- Complexity $2^{n-n1}$
  - Set $n_1 \approx n$ ?
- Parity computations cost $\binom{n-n_1}{\deg(U)} \cdot 2^{n_1}$
- Set $n_1$ to balance complexity

# Plan

- Background
- Overview of the algorithm
- **Conclusion and open problems**

# Conclusions and Open Problems

- Described **concretely efficient** polynomial method algorithm for solving (random) $\mathbb{F}_2$ equations
  - Complexity: $n^2 \cdot 2^{0.815n}$ for $d=2$, $n^2 \cdot 2^{(1-2.7d)n}$ for $d \geq 2$
- In paper:
  - Reduce **memory complexity** by exponential factor (still high)
  - Optimize algorithm for **concrete cryptosystems** (Picnic \ LowMC)

- Open problems:
  - Can algorithm be improved ?
  - Can algorithm be optimized for **over-defined** systems $n \ll m$ ?
  - **Fast implementation** of (variant of) algorithm ?

# Thanks for your attention!