Multivariate Quadratic Public-Key Cryptography Part 3: Small Field Schemes

Bo-Yin Yang

Academia Sinica

PQCrypto Executive Summer School 2017
Eindhoven, the Netherlands
Friday, 23.06.2017

Oil-Vinegar Polynomials [Patarin 1997]

Let $\mathbb{F}$ be a (finite) field. For $o, v \in \mathbb{N}$ set $n = o + v$ and define

$$p(x_1, \ldots, x_n) = \underbrace{\sum_{i=1}^{v} \sum_{j=i}^{v} \alpha_{ij} \cdot x_i \cdot x_j}_{v \times v \text{ terms}} + \underbrace{\sum_{i=1}^{v} \sum_{j=v+1}^{n} \beta_{ij} \cdot x_i \cdot x_j}_{v \times o \text{ terms}} + \underbrace{\sum_{i=1}^{n} \gamma_i \cdot x_i}_{\text{linear terms}} + \delta$$

$x_1, \ldots, x_v$: Vinegar variables $x_{v+1}, \ldots, x_n$: Oil variables, no $o \times o$ terms.

If we randomly set $x_1, \ldots, x_v$, result is linear in $x_{v+1}, \ldots, x_n$

(Unbalanced) Oil-Vinegar matrix

$\tilde{p}$ the homogeneous quadratic part of $p(x_1, \ldots, x_n)$ can be written as quadratic form $\tilde{p}(\mathbf{x}) = \mathbf{x}^T \cdot M \cdot \mathbf{x}$ with

$$M = \left(\begin{array}{c|c} *_{v \times v} & *_{o \times v} \\ \hline *_{v \times o} & 0_{o \times o} \end{array} \right)$$

where $*$ denotes arbitrary entries subject to symmetry.

Inversion of the UOV central map

Let each of o components of a UOV central map be a UOV polynomial.

After guessing Vinegar variables

When we guess the Vinegar variables $x_1, \ldots, x_v$, we get o linear equations in the o Oil variables $x_{v+1}, \ldots, x_n \Rightarrow$ recovered by (Gaussian) elimination

If the system has no solution?

Just choose other values for the Vinegar variables $x_1, \ldots, x_v$ and try again.

Inversion of the UOV central map

Let each of o components of a UOV central map be a UOV polynomial.

After guessing Vinegar variables

When we guess the Vinegar variables $x_1, \ldots, x_v$, we get o linear equations in the o Oil variables $x_{v+1}, \ldots, x_n \Rightarrow$ recovered by (Gaussian) elimination

Toy Example in $\mathbb{F} = \mathrm{GF}(7)$ with $o = v = 2$

- $\mathcal{Q} = (f^{(1)}, f^{(2)})$ with

$$f^{(1)}(\mathbf{x}) = 2x_1^2 + 3x_1x_2 + 6x_1x_3 + x_1x_4 + 4x_2^2 + 5x_2x_4 + 3x_1 + 2x_2 + 5x_3 + x_4 + 6,$$
$$f^{(2)}(\mathbf{x}) = 3x_1^2 + 6x_1x_2 + 5x_1x_4 + 3x_2^2 + 5x_2x_3 + x_2x_4 + 2x_1 + 5x_2 + 4x_3 + 2x_4 + 1.$$

- Goal: Find a pre image $\mathcal{Q}^{-1}(\mathbf{y})$, $\mathbf{y} = (3, 4)$
- Choose random values for x_1 and x_2, e.g. $(x_1, x_2) = (1, 4)$
- $\tilde{f}^{(1)}(x_3, x_4) = 4x_3 + x_4 + 4 = w_1 = 3$, $\tilde{f}^{(2)}(x_3, x_4) = 3x_3 + 4x_4 = w_2 = 4$
- The pre image of $\mathbf{y}$ is $\mathbf{x} = (1, 4, 1, 2)$.

Operations of UOV

Key Generation

Take a UOV central map $\mathcal{Q}$ and invertible $\mathcal{S} : \mathbb{F}^n \to \mathbb{F}^n$. $\mathcal{P} = \mathcal{Q} \circ \mathcal{S}$.

Signature Generation

1. Given: message d, take its hash $\mathbf{y} = \mathcal{H}(d)$ under $\mathcal{H} : \{0,1\}^\star \to \mathbb{F}^o$.
2. Compute a pre-image $\mathbf{x} \in \mathbb{F}^n$ of $\mathbf{y}$ under the central map $\mathcal{Q}$
    - Choose random values for the Vinegar variables $x_1, \ldots, x_v$ and substitute them into the central map polynomials $f^{(1)}, \ldots, f^{(o)}$
    - Solve the resulting linear system for the Oil variables $x_{v+1}, \ldots, x_n$
    - If the system has no solution, choose other values for the Vinegar variables and try again.
3. Compute the signature $\mathbf{w} \in \mathbb{F}^n$ by $\mathbf{w} = \mathcal{S}^{-1}(\mathbf{x})$.

Operations of UOV

Key Generation

Take a UOV central map $\mathcal{Q}$ and invertible $\mathcal{S} : \mathbb{F}^n \to \mathbb{F}^n$. $\mathcal{P} = \mathcal{Q} \circ \mathcal{S}$.

Signature Generation

1. Given: message d, take its hash $\mathbf{y} = \mathcal{H}(d)$ under $\mathcal{H} : \{0,1\}^\star \to \mathbb{F}^o$.
2. Compute a pre-image $\mathbf{x} \in \mathbb{F}^n$ of $\mathbf{y}$ under the central map $\mathcal{Q}$
3. Compute the signature $\mathbf{w} \in \mathbb{F}^n$ by $\mathbf{w} = \mathcal{S}^{-1}(\mathbf{x})$.

Signature Verification

Given: message d, signature $\mathbf{w} \in \mathbb{F}^n$

1. Compute $\mathbf{z} = \mathcal{H}(d)$.
2. Compute $\mathbf{z}' = \mathcal{P}(\mathbf{w})$.

Accept the signature $\Leftrightarrow \mathbf{z} = \mathbf{z}'$

Kipnis-Shamir OV attack when $o = v$

$$\begin{aligned} \mathcal{O} &:= \{\mathbf{x} \in \mathbb{F}^n : x_1 = \ldots = x_v = 0\} \quad \text{“Oilspace”} \\ \mathcal{V} &:= \{\mathbf{x} \in \mathbb{F}^n : x_{v+1} = \ldots = x_n = 0\} \quad \text{“Vinegarspace”} \end{aligned}$$

Let E, F be invertible “OV-matrices”, i.e. $E, F = \begin{pmatrix} \star & \star \\ \star & 0 \end{pmatrix}$ Then

$E \cdot \mathcal{O} \subset \mathcal{V}$. Since the two has the same rank, equality holds, so $(F^{-1} \cdot E) \cdot \mathcal{O} = \mathcal{O}$, i.e. $\mathcal{O}$ is an invariant subspace of $F^{-1} \cdot E$.

Kipnis-Shamir OV attack when $o = v$

$$\mathcal{O} := \{\mathbf{x} \in \mathbb{F}^n : x_1 = \ldots = x_v = 0\} \quad \text{"Oilspace"}$$
$$\mathcal{V} := \{\mathbf{x} \in \mathbb{F}^n : x_{v+1} = \ldots = x_n = 0\} \quad \text{"Vinegarspace"}$$

Let E, F be invertible "OV-matrices", i.e. $E, F = \begin{pmatrix} \star & \star \\ \star & 0 \end{pmatrix}$ Then

$E \cdot \mathcal{O} \subset \mathcal{V}$. Since the two has the same rank, equality holds, so $(F^{-1} \cdot E) \cdot \mathcal{O} = \mathcal{O}$, i.e. $\mathcal{O}$ is an invariant subspace of $F^{-1} \cdot E$.

Common Subspaces

Let H_i be the matrix representing the homogeneous quadratic part of the i-th public polynomial. Then we have $H_i = S^T \cdot E_i \cdot S$, i.e. $TS^{-1}(\mathcal{O})$ is an invariant subspace of the matrix $(H_j^{-1} \cdot H_i)$, and we find S^{-1}.

Kipnis-Shamir OV attack when $o = v$

$$\begin{array}{rcll} \mathcal{O} & := & \{\mathbf{x} \in \mathbb{F}^n : x_1 = \ldots = x_v = 0\} & \text{“Oilspace”} \\ \mathcal{V} & := & \{\mathbf{x} \in \mathbb{F}^n : x_{v+1} = \ldots = x_n = 0\} & \text{“Vinegarspace”} \end{array}$$

Let E, F be invertible “OV-matrices”, i.e. $E, F = \begin{pmatrix} \star & \star \\ \star & 0 \end{pmatrix}$ Then $E \cdot \mathcal{O} \subset \mathcal{V}$. Since the two has the same rank, equality holds, so $(F^{-1} \cdot E) \cdot \mathcal{O} = \mathcal{O}$, i.e. $\mathcal{O}$ is an invariant subspace of $F^{-1} \cdot E$.

Common Subspaces

Let H_i be the matrix representing the homogeneous quadratic part of the i-th public polynomial. Then we have $H_i = S^T \cdot E_i \cdot S$, i.e. $TS^{-1}(\mathcal{O})$ is an invariant subspace of the matrix $(H_j^{-1} \cdot H_i)$, and we find S^{-1}.

Summary of the Standard UOV Attack

- for $v \le o$, breaks the balanced OV scheme in polynomial time.
- For $v > o$ the complexity of the attack is about $q^{v-o} \cdot o^4$.

$\Rightarrow$ Choose $v \approx 2 \cdot o$ (unbalanced Oil and Vinegar (UOV)) [KP99]

Other Attacks

- **Collision Attack**: $o \geq \frac{2^{2\ell}}{\log_2(q)}$ for ℓ-bit security.
- **Direct Attack**: Try to solve the public equation $\mathcal{P}(\mathbf{w}) = \mathbf{z}$ as an instance of the MQ-Problem. The public systems of UOV behave much like random systems, but they are highly underdetermined ($n = 3 \cdot m$)

  Result [Thomae]: A multivariate system of m equations in $n = \omega \cdot m$ variables can be solved in the same time as a determined system of $m - \lfloor \omega \rfloor + 1$ equations.

  $\Rightarrow$ m has to be increased by 2.

Other Attacks

- **Collision Attack**: $o \geq \frac{2^{2\ell}}{\log_2(q)}$ for ℓ-bit security.
- **Direct Attack**: Try to solve the public equation $\mathcal{P}(\mathbf{w}) = \mathbf{z}$ as an instance of the MQ-Problem. The public systems of UOV behave much like random systems, but they are highly underdetermined $(n = 3 \cdot m) \;\Rightarrow\; m$ has to be increased by 2.
- **UOV-Reconciliation attack**: Try to find a linear transformation $\mathcal{S}$ ("good keys") which transforms the public matrices H_i into the form of UOV matrices

$$(S^T)^{-1} \cdot H_i \cdot S^{-1} = \begin{pmatrix} \star & \star \\ \star & 0 \end{pmatrix}, \qquad S = \begin{pmatrix} 1 & \star \\ 0 & 1 \end{pmatrix}$$

$\Rightarrow$ Each Zero-term yields a quadratic equation in the elements of T.
$\Rightarrow$ T can be recovered by solving several MQ systems (the hardest with v variables, m equations).

Summary of UOV

Safe Parameters for UOV($\mathbb{F}$, o, v)

security level (bit)	scheme	public key size (kB)	private key size (kB)	hash size (bit)	signature (bit)
80	UOV($\mathbb{F}_{16}$,40,80)	144.2	135.2	160	480
	UOV($\mathbb{F}_{256}$,27,54)	89.8	86.2	216	648
100	UOV($\mathbb{F}_{16}$,50,100)	280.2	260.1	200	600
	UOV($\mathbb{F}_{256}$, 34,68)	177.8	168.3	272	816
128	UOV($\mathbb{F}_{16}$,64,128)	585.1	538.1	256	768
	UOV($\mathbb{F}_{256}$,45,90)	409.4	381.8	360	1,080
192	UOV($\mathbb{F}_{16}$,96,192)	1,964.3	1,786.7	384	1,152
	UOV($\mathbb{F}_{256}$,69,138)	1,464.6	1,344.0	552	1,656
256	UOV($\mathbb{F}_{16}$,128,256)	4,644.1	4,200.3	512	1,536
	UOV($\mathbb{F}_{256}$,93,186)	3,572.9	3,252.2	744	2,232

What we know today about UOV

- unbroken since 1999 $\Rightarrow$ high confidence in security
- not the fastest multivariate scheme
- very large keys, (comparably) large signatures

Rainbow Digital Signature

Ding and Schmidt, 2004

- Patented by Ding
- May have had patent by T.-T. Moh (expired)
- TTS is its variant with sparse central map

Rainbow Digital Signature

Ding and Schmidt, 2004

- Finite field $\mathbb{F}$, integers $0 < v_1 < \cdots < v_u < v_{u+1} = n$.
- Set $V_i = \{1, \ldots, v_i\}$, $O_i = \{v_i + 1, \ldots, v_{i+1}\}$, $o_i = v_{i+1} - v_i$.
- Central map $\mathcal{Q}$ consists of $m = n - v_1$ polynomials $f^{v_1+1}, \ldots, f^{(n)}$ of the form

$$f^{(k)} = \sum_{i,j \in V_\ell} \alpha_{ij}^{(k)} x_i x_j + \sum_{i \in V_\ell, j \in O_\ell} \beta_{ij}^{(k)} x_i x_j + \sum_{i \in V_\ell \cup O_\ell} \gamma_i^{(k)} x_i + \delta^{(k)},$$

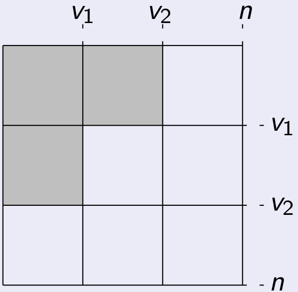
  with coefficients $\alpha_{ij}^{(k)}$, $\beta_{ij}^{(k)}$, $\gamma_i^{(k)}$ and $\delta^{(k)}$ randomly chosen from $\mathbb{F}$ and ℓ being the only integer such that $k \in O_\ell$.
- Choose randomly two affine (or linear) transformations $\mathcal{T} : \mathbb{F}^m \to \mathbb{F}^m$ and $\mathcal{S} : \mathbb{F}^n \to \mathbb{F}^n$.
- *public key*: $\mathcal{P} = \mathcal{T} \circ \mathcal{Q} \circ \mathcal{S} : \mathbb{F}^n \to \mathbb{F}^m$
- *private key*: $\mathcal{T}$, $\mathcal{Q}$, $\mathcal{S}$

Idea of Rainbow

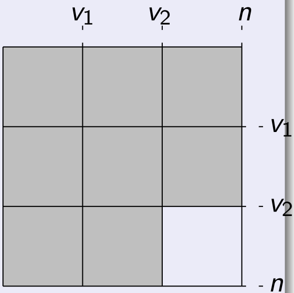
Inversion of the central map

- Invert the single UOV layers recursively.
- Use the variables of the i-th layer as Vinegars of the $i+1$-th layer.

Illustration: Rainbow with two layers

$F^{(k)} =$ 

$v_1 + 1 \leq k \leq v_2$

$F^{(k)} =$ 

$v_2 + 1 \leq k \leq n$

Idea of Rainbow

Inversion of the central map

- Invert the single UOV layers recursively.
- Use the variables of the i-th layer as Vinegars of the $i+1$-th layer.

Input: Rainbow central map $\mathcal{Q} = (f^{(v_1+1)}, \ldots, f^{(n)})$, vector $\mathbf{y} \in \mathbb{F}^m$.
Output: vector $\mathbf{x} \in \mathbb{F}^n$ with $\mathcal{Q}(\mathbf{x}) = \mathbf{y}$.

1: Choose random values for the variables $x_1, \ldots, x_{v_1}$ and substitute these values into the polynomials $f^{(i)}$ $(i = v_1 + 1, \ldots n)$.
2: **for** $\ell = 1$ to u **do**
3: Perform Gaussian Elimination on the polynomials $f^{(i)}$ $(i \in O_\ell)$ to get the values of the variables x_i $(i \in O_\ell)$.
4: Substitute the values of x_i $(i \in O_\ell)$ into the polynomials $f^{(i)}$ $(i = v_{\ell+1} + 1, \ldots, n)$.
5: **end for**

Idea of Rainbow

Inversion of the central map

- Invert the single UOV layers recursively.
- Use the variables of the i-th layer as Vinegars of the $i+1$-th layer.

Signature Generation from message d

1. Use a hash function $\mathcal{H} : \{0,1\} \to \mathbb{F}^m$ to compute $\mathbf{z} = \mathcal{H}(d) \in \mathbb{F}^m$
2. Compute $\mathbf{y} = \mathcal{T}^{-1}(\mathbf{z}) \in \mathbb{F}^m$.
3. Compute a pre-image $\mathbf{x} \in \mathbb{F}^n$ of $\mathbf{y}$ under the central map $\mathcal{Q}$
4. Compute the signature $\mathbf{w} \in \mathbb{F}^n$ by $\mathbf{w} = \mathcal{S}^{-1}(\mathbf{x})$.

Idea of Rainbow

Inversion of the central map

- Invert the single UOV layers recursively.
- Use the variables of the i-th layer as Vinegars of the $i+1$-th layer.

Signature Generation from message d

1. Use a hash function $\mathcal{H} : \{0,1\} \to \mathbb{F}^m$ to compute $\mathbf{z} = \mathcal{H}(d) \in \mathbb{F}^m$
2. Compute $\mathbf{y} = \mathcal{T}^{-1}(\mathbf{z}) \in \mathbb{F}^m$.
3. Compute a pre-image $\mathbf{x} \in \mathbb{F}^n$ of $\mathbf{y}$ under the central map $\mathcal{Q}$
4. Compute the signature $\mathbf{w} \in \mathbb{F}^n$ by $\mathbf{w} = \mathcal{S}^{-1}(\mathbf{x})$.

Signature Verification from message d, signature $\mathbf{z} \in \mathbb{F}^n$

1. Compute $\mathbf{z} = \mathcal{H}(d)$.
2. Compute $\mathbf{z}' = \mathcal{P}(\mathbf{w})$.

Accept the signature $\mathbf{z} \Leftrightarrow \mathbf{w}' = \mathbf{w}$.

Security

Rainbow is an extension of UOV
$\Rightarrow$ All attacks against UOV can be used against Rainbow, too.

Additional structure of the central map allows several new attacks

- **MinRank Attack**: Look for linear combinations of the matrices H_i of low rank
- **HighRank Attack**: Look for the linear representation of the variables appearing the lowest number of times in the central polynomials.
- **Rainbow-Band-Separation Attack**: Variant of the UOV-Reconciliation Attack using the additional Rainbow structure [DY08]

Choosing Parameter Selection for Rainbow is interesting

MinRank Attack

Minors Version

Set all rank $r+1$ minors of $\sum_i \alpha_i H_i$ to 0.

Kernel Vector Guessing Version

- Guess a vector $\mathbf{v}$, let $\sum_i \alpha_i H_i \mathbf{v} = 0$, hope to find a non-trivial solution.
- (If $m > n$, guess $\lceil \frac{m}{n} \rceil$ vectors.)
- Takes $q^r m^3/3$ time to find a r-dimensional subspace.

Accumulation of Kernels and Effective Rank

In the first stage of Rainbow, there are $o_1 = v_2 - v_1$ equations and v_2 variables. The rank should be v_2. But if your guess corresponds to $x_1 = x_2 = \cdots = x_{v_1} = 0$, then about $1/q$ of the time we find a kernel. The easy way to see this is that there are $q^{o_1 - 1}$ different kernels. We say that "effectively the rank is $v_1 + 1$".

Rainbow Band Separation

Extension to UOV reconciliation to use the special Rainbow form.

n variables, $n+m-1$ quadratic equations

1. Let $w_i := w'_i - \lambda_i w'_n$ for $i \le v$, $w_i = w'_i$ for $i > v$. Evaluate $\mathbf{z}$ in $\mathbf{w}'$.
2. Find m equations by letting all $(w'_n)^2$ terms vanish; there are v of λ_i's.
3. Set all cross-terms involving w'_n in $z_1 - \sigma_1^{(1)} z_{v+1} - \sigma_2^{(1)} z_{v+2} - \cdots - \sigma_o^{(1)} z_m$ to be zero and find $n-1$ more equations.
4. Solve $m+n-1$ quadratic equations in $o+v=n$ unknowns.
5. Repeat, e.g. next set $w'_i := w''_i - \lambda_i w''_{n-1}$ for $i < v$, and let every $(w''_{n-1})^2$ and $w''_n w''_{n-1}$ term be 0. Also set $z_2 - \sigma_1^{(2)} z_{v+1} - \sigma_2^{(2)} z_{v+2} - \cdots - \sigma_o^{(2)} z_m$ to have a zero second-to-last column. [$2m+n-2$ equations in n unknowns.]

Rainbow - Summary

- no weaknesses found since 2007
- efficient, much faster than RSA
- suitable for low cost devices
- shorter signatures and smaller key sizes than UOV

Parameters

security level (bit)	parameters $\mathbb{F}, v_1, o_1, o_2$	public key size (kB)	private key size (kB)	hash size (bit)	signature (bit)
80	$\mathbb{F}_{16}$,20,20,20	33.4	22.3	160	228
	$\mathbb{F}_{256}$,19,12,13	25.3	19.3	200	352
100	$\mathbb{F}_{16}$,25,25,25	65.9	43.2	200	288
	$\mathbb{F}_{256}$, 27,16,16	57.2	44.3	256	472
128	$\mathbb{F}_{16}$,32,32,32	136.6	87.6	256	368
	$\mathbb{F}_{31}$,28,28,28	123.2	74.5	280	420
	$\mathbb{F}_{256}$,36,21,22	136.0	102.5	344	632
192	$\mathbb{F}_{16}$,48,48,48	475.9	301.8	384	564
	$\mathbb{F}_{31}$,44,40,40	360.1	245.2	420	630
256	$\mathbb{F}_{16}$,64,64,64	1,194.4	763.9	512	776

References

Pa97 J. Patarin: The oil and vinegar signature scheme, presented at the Dagstuhl Workshop on Cryptography (September 97)

KS98 A. Kipnis, A. Shamir: Cryptanalysis of the Oil and Vinegar Signature scheme. CRYPTO 1998, LNCS vol. 1462, pp. 257–266. Springer, 1988.

KP99 A. Kipnis, J. Patarin, L. Goubin: Unbalanced Oil and Vinegar Schemes. EUROCRYPT 1999. LNCS vol. 1592, pp. 206–222 Springer, 1999.

DS05 J. Ding, S. Schmidt: Rainbow, a new multivariate polynomial signature scheme. ACNS 2005. LNCS vol. 3531, pp. 164–175 Springer, 2005.

DY08 J. Ding, B.Y. Yang, C.H.O. Chen, M.S. Chen, C.M. Cheng: New Differential-Algebraic Attacks and Reparametrization of Rainbow. ACNS 2008, LNCS 5037, pp.242–257, Springer 2008.

Multivariates Part 4: Implementation on Modern CPUs

Some Lessons from the Last 14 Years

Bo-Yin Yang

Academia Sinica

PQCrypto Executive Summer School 2017
Eindhoven, the Netherlands
Friday, 23.06.2017

Why are MPKCs Worth Studying?

- Diversification
- Efficiency

Why are MPKCs Worth Studying?

- Diversification: Future-proof against quantum computers.
- Efficiency: Faster than “traditional” PKCs.

Why are MPKCs Worth Studying?

- Diversification: Future-proof against quantum computers.
- Efficiency: Faster than "traditional" PKCs.
  … Maybe.

Rate-Determining Mechanisms for MPKCs

Key Generation

Evaluation of coefficients

Public Maps

Evaluating a generic set of quadratic polynomials in $\mathbb{K} = \mathbb{F}_q$

Private Maps

Rate-Determining Mechanisms for MPKCs

Key Generation

Evaluation of coefficients:

- Often as differentials of public map.
- Sometimes, by brute force!

Public Maps

Evaluating a generic set of quadratic polynomials in $\mathbb{K} = \mathbb{F}_q$

Private Maps

Rate-Determining Mechanisms for MPKCs

Key Generation

Evaluation of coefficients

Public Maps

Evaluating a generic set of quadratic polynomials in $\mathbb{K} = \mathbb{F}_q$
usually as a matrix multiplying the vector of monomials

Private Maps

Rate-Determining Mechanisms for MPKCs

Key Generation

Evaluation of coefficients

Public Maps

Evaluating a generic set of quadratic polynomials in $\mathbb{K} = \mathbb{F}_q$

Private Maps

UOV	Solving linear systems of equations in $\mathbb{K} = \mathbb{F}_q$
Rainbow	Like UOV plus mini "Public Map"
C^*	High powers in $\mathbb{L} = \mathbb{F}_{q^n}$
HFE	Equation solving in $\mathbb{L} = \mathbb{F}_{q^n}$ (general arithmetic)
kHFE	Like HFE plus an elimination in $\mathbb{L}$

Practical Side of Computing

Moore's law

Transistor budget doubles every 18–24 months

Memory Latencies vs Clock Speeds

Year	Hi-End CPU	MHz	DRAM
1979	Z80	2	500ns
1984	80286	10	400ns
1989	80486	40	300ns
1994	Pentium	100	250ns
1999	Athlon	750	200ns
2004	Pentium 4	3800	160ns
2009	Core i7	*3200*	130ns
2014	Core i7	*3400*	120ns

Are MPKCs Still Fast?

- Progress in high-precision arithmetic
  - In 80's, CPUs computed one 32-bit integer product every 15–20 cycles
  - In 2000, x86 CPUs computed one 64-bit product every 3–10 cycles
  - Core i7's today produces one 128-bit product every 1 cycle
  - Marvelous for ECC (and RSA)
- In contrast, progress in $\mathbb{F}_{2^q}$ arithmetic is *slow*
  - 6502 or 8051: a dozen cycles via three table look-ups
  - Modern x86: roughly same that many cycles
- Moore's law favors computation, not so much memories
  - Memory access speed increased at a snail's pace
- Wang et al. made life even harder for MPKCs
  - Forcing longer message digests
  - RSA untouched

Questions We Want to Answer

- Can all the extras on modern commodity CPUs help MPKCs as well?
- How have architectural changes affected implementation choices?
- If so, how do MPKCs compare to traditional PKCs today?

SSE, the X86 Vector Instruction Set Extensions

- As packed 8-, 16-, 32- or 64-bit operands
- Move `xmm` to/from `xmm`, memory (even unaligned), x86 registers
- Shuffle data and pack/unpack on vector data
- Bit-wise logical operations like AND, OR, NOT, XOR
- Shift left, right logical/arithmetic by units, or entire `xmm` byte-wise
- Add/subtract on 8-, 16-, 32- and 64-bits
- Multiply 16-bit and 32-bits in various ways
- VPSHUFB (32 nibble-to-byte lookup in 1 cycle) and PALIGNR (256-bit bytewise rotation) quite powerful

(V)PSHUFB

- "Packed Shuffle Bytes"
  - Source: $(x_0, \ldots, x_{15})$
  - Destination: $(y_0, \ldots, y_{15})$
  - Result: $(y_{x_0 \bmod 32}, \ldots, y_{x_{15} \bmod 32})$, treating $y_{16}, \ldots, y_{31}$ as 0
- VPSHUFB = two individual PSHUFBs

Speeding Up MPKCs over $\mathbb{F}_{16}$

- TT : 16×16 table, with $TT_{i,j} = i * j, 0 \leq i, j < 16$
- To compute $a\mathbf{v}$, $a \in \mathbb{F}_{16}, \mathbf{v} \in (\mathbb{F}_{16})^{16}$
  - `xmm` $\leftarrow$ a-th row of TT
  - $a\mathbf{v} \leftarrow$ PSHUFB `xmm`,$\mathbf{v}$
- Works similarly for $\mathbf{a} \in (\mathbb{F}_{16})^2, \mathbf{v} \in (\mathbb{F}_{16})^{32}$
  - Need to unpack, do PSHUFBs, then pack
- Delivers $2\times$ performance over simple bit slicing in private map evaluation of rainbow and TTS
- Some other platforms also have similar instructions
  - AMD's SSE5: PPERM (superset of PSHUFB)
  - IBM POWER AltiVec/VMX: PERMU
  - ARM's TBL

Speeding Up MPKCs over $\mathbb{F}_{256}$

Nibble Slicing

- TL : 256×16 table, with $TL_{i,j} = i * j, 0 \le i < 256,\ 0 \le j < 16$
- TH : 256×16 table, with $TH_{i,j} = i * (16j), 0 \le i < 256,\ 0 \le j < 16$
- To compute $a\mathbf{v}$, $a \in \mathbb{F}_{256}, \mathbf{v} \in (\mathbb{F}_{256})^{16}$
  - $a\mathbf{v}_i = a(16\lfloor \mathbf{v}_i/16 \rfloor) + a(\mathbf{v}_i \bmod 16), 0 \le i < 16$
- $\mathbf{v}'_i \leftarrow a(16\lfloor \mathbf{v}_i/16 \rfloor)$
  - $\mathbf{v}'_i \leftarrow \lfloor \mathbf{v}_i/16 \rfloor$ (SHIFT)
  - `xmm` $\leftarrow$ a-th row of TH
  - $\mathbf{v}' \leftarrow$ PSHUFB `xmm`,$\mathbf{v}'$
- $\mathbf{v}_i \leftarrow a(\mathbf{v}_i \bmod 16)$
  - $\mathbf{v}_i \leftarrow \mathbf{v}_i \bmod 16$ (AND)
  - `xmm` $\leftarrow$ a-th row of TL
  - $\mathbf{v} \leftarrow$ PSHUFB `xmm`,$\mathbf{v}$
- $a\mathbf{v} \leftarrow \mathbf{v} + \mathbf{v}'$ (OR)

Arithmetic in $\mathbb{F}_{2^k}$

PCLMULQDQ

Of course you use it if you can, sheesh.

Multiplication Tables in Memory (Parallel)

Log/Exp Tables to a generator g

Bit-Slicing

Arithmetic in $\mathbb{F}_{2^k}$

PCLMULQDQ

Of course you use it if you can, sheesh.

Multiplication Tables in Memory (Parallel)

- One VPSHUFB per many multiplications in $\mathbb{F}_{16}$
- How do we do time-constant Table Lookups?

Log/Exp Tables to a generator g

Bit-Slicing

Arithmetic in $\mathbb{F}_{2^k}$

PCLMULQDQ

Of course you use it if you can, sheesh.

Multiplication Tables in Memory (Parallel)

Log/Exp Tables to a generator g

- Compute xy as $g^{\log_g x + \log_g y}$ if neither is zero.
- 3 lookups per mult, some logs can be pre-computed
- Time-constant but method of last choice.

Bit-Slicing

Arithmetic in $\mathbb{F}_{2^k}$

PCLMULQDQ

Of course you use it if you can, sheesh.

Multiplication Tables in Memory (Parallel)

Log/Exp Tables to a generator g

Bit-Slicing

- Highly parallel — 32/64/128 multiplies at the same time
- Often requires rearranging of data
- Parameters can result in awkward dimensions like $1 +$ (word size)
- only good for $\mathbb{F}_2$ and $\mathbb{F}_4$.

Arithmetic in $\mathbb{F}_{2^k}$

PCLMULQDQ

Of course you use it if you can, sheesh.

Multiplication Tables in Memory (Parallel)

For time-constancy, we can build Multiplication Tables on the Fly.

Log/Exp Tables to a generator g

Bit-Slicing

- Highly parallel — 32/64/128 multiplies at the same time
- Often requires rearranging of data
- Parameters can result in awkward dimensions like $1 + (\text{word size})$
- only good for $\mathbb{F}_2$ and $\mathbb{F}_4$.

Some Interesting Design Choices

System and Architecture-Dependent Stuff

- Key Generation
- Matrix-to-Vector-Multiply and Evaluating Public Maps
- Tower Field Arithmetic
- System- and Equation-Solving
  - Pre-scripted Gröbner Basis Computation
  - Iterative Methods vs. Gaussian Eliminations
  - Cantor-Zassenhaus vs. Berlekamp

Key Generation

Matsumoto-Imai's notaton: $z_k := \sum_i w_i \left[P_{ik} + Q_{ik} w_i + \sum_{j<i} R_{ijk} w_j \right].$

Usual Way: as differentials of public map $\mathcal{P} = (p_1, \ldots, p_m)$

for $q > 2$, we choose any $a \neq 0, 1$ and get

$$
\begin{aligned}
Q_{ik} &:= (a(a-1))^{-1} (p_k(a\mathbf{v}_i) - a p_k(\mathbf{v}_i)) \\
P_{ik} &:= p_k(\mathbf{v}_i) - Q_{ik} \\
R_{ijk} &:= p_k(\mathbf{v}_i + \mathbf{v}_j) - Q_{ik} - Q_{jk} - P_{ik} - P_{jk}
\end{aligned}
$$

For $\mathbb{F}_2$, it becomes

$$
\begin{aligned}
P_{ik} &:= p_k(\mathbf{v}_i) \\
R_{ijk} &:= p_k(\mathbf{v}_i + \mathbf{v}_j) - P_{ik} - P_{jk}
\end{aligned}
$$

($\mathbf{v}_i$ means the unit vector on the i-th direction)

Evaluating Public Maps

Naive Way (and on μP's still)

$$z_k = \sum_i w_i \left[P_{ik} + Q_{ik} w_i + \sum_{i<j} R_{ijk} w_j \right]$$

For better memory access pattern

1. $\mathbf{c} \leftarrow [\mathbf{w}^T, (w_i w_j)_{i \le j}]^T$
2. $\mathbf{z} \leftarrow \mathbf{P}\mathbf{c}$, where $\mathbf{P}$ is the $m \times n(n+3)/2$ public-key matrix

How to do Matrix-to-Vector mults

Microcontrollers Naively

Somewhat newer CPUs Bit-slicing for $\mathbb{F}_{2^k}$

With more cache Big look-up tables (with nibble-slicing)

Newest architectures More or less naively, with SSE*

MPKCs over Odd Prime Fields

MPKCs over Odd Prime Fields

Are you out of your mind?

- XOR is easy, addition mod q is not.
- How can it possibly be faster?

MPKCs over Odd Prime Fields

Are you out of your mind?

- XOR is easy, addition mod q is not.
- How can it possibly be faster?

It's more than about speed

- Good for defending against Gröbner basis attacks
  - The field equation $X^q - X = 0$ becomes much less useful
- SSE* gives you parallel arithmetic on small integers,
  - and you only need to parallelize 4 or 8 at a time.
- Do you know how many 18-bit multipliers there are on an FPGA?

Basic Building Blocks for Speeding Up Odd MPKCs

PMULHRSW

takes upper half in a rounded signed product of two 16-bit words, $\lceil xy/2^{16} \rfloor$, good for reduction mod q

VPMADDUSBW

- Packed Multiply and Add, Unsigned and Signed Byte to Word
  - Source: $(x_0, \ldots, x_{31})$ Unsigned
  - Destination: $(y_0, \ldots, y_{31})$ Signed
  - Result: $(x_0 y_0 + x_1 y_1, x_2 y_2 + x_3 y_3, \ldots, x_{30} y_{30} + x_{31} y_{31})$
- Helpful in evaluating $\mathbf{z} = \mathbf{Pc}$, piece by piece
  - Let $\mathbf{Q}$ be a 16×2 submatrix of $\mathbf{P}$
  - $\mathbf{d}^T$ be the corresponding 2×1 submatrix of $\mathbf{c}$
  - r1 $\leftarrow (Q_{11}, Q_{12}, Q_{21}, Q_{22}, \ldots, Q_{15,1}, Q_{15,2})$
  - r2 $\leftarrow (d_1, d_2, d_1, d_2, \ldots, d_1, d_2)$
  - VPMADDUSBW r1, r2 computes $\mathbf{Qd}$
  - Continue in 16-bits until reduction $\bmod q$ needed.
- Saves a few $\bmod q$ operations and delivers $1.5\times$ performance

Big look-up tables for matrix multiplication

As suggested by Berbain *et al*, SAC 2006

- Pre-compute $a\mathbf{v}$ for each column $\mathbf{v}$ in any constant matrix
- Read off the appropriately offset vector as needed
- Can nibble-slice $\mathbb{F}_{16}/\mathbb{F}_{256}$ into $\mathbb{F}_{16}/\mathbb{F}_4$
- Obviously minimizes the need for operations

Big look-up tables for matrix multiplication

As suggested by Berbain *et al*, SAC 2006

- Pre-compute $a\mathbf{v}$ for each column $\mathbf{v}$ in any constant matrix
- Read off the appropriately offset vector as needed
- Can nibble-slice $\mathbb{F}_{16}/\mathbb{F}_{256}$ into $\mathbb{F}_{16}/\mathbb{F}_4$
- Obviously minimizes the need for operations

Unbelievably ...

Slower than SSE on any modern CPU!

Big look-up tables for matrix multiplication

As suggested by Berbain *et al*, SAC 2006

- Pre-compute $a\mathbf{v}$ for each column $\mathbf{v}$ in any constant matrix
- Read off the appropriately offset vector as needed
- Can nibble-slice $\mathbb{F}_{16}/\mathbb{F}_{256}$ into $\mathbb{F}_{16}/\mathbb{F}_4$
- Obviously minimizes the need for operations

Unbelievably ...

Slower than SSE on any modern CPU!

When L2 isn't fast enough

- SSE instructions have a reverse throughput of 1 cycle today
- memory access is linear when using SSE
- L2 latency 20+ cycles; LUT reads not regular enough
- No way around this today :(

Remarks on Getting More Performance

Laziness often leads to optimality

- Do not always need the tightest range
- The less reductions, the better!
- The less memory access, the better!
- The more regular memory access, the better!
- Packing $\mathbb{F}_q$-blocks into binary can use more bits than necessary as long as the map is injective and convenient to compute

Wiedemann vs. Gauss Elimination mod q

- How to solve a medium-sized dense linear system?
  - Wiedemann iterative solver for $\mathbf{Ax} = \mathbf{b}$
    - Compute $\mathbf{zA}^i\mathbf{b}$ for some $\mathbf{z}$
    - Compute minimal polynomial using Berlekamp-Massey
  - Requires $O(2n^3)$ field multiplications
  - Straightforward Gauss elimination requires $O(n^3/3)$
- However, Wiedemann involves much less reductions modulo q
- However, everything has to be constant-time
- At the moment Gaussian beats Wiedemann.

To Solve Equation(s) in a Big Tower Field over $\mathbb{F}_q$

Scripted Gröbner Basis Computation

From 3 quadratic equations in 3 variables, we in succession run Gaussian eliminations on matrices of dimensions 3×10, 11×19, 8×16, 5×13, with many coefficients that we know to be zero in advance, to reach a degree-8 equation. You can call this a tailored matrix-$\mathbf{F_4}$.

Cantor-Zassenhaus (instead of Berlekamp)

1. Replace $u(X)$ by $\gcd(u(X), X^{q^k} - X)$ so that u splits in $\mathbb{L}$.
   1. Compute and tabulate $X^d \bmod u(X), \ldots, X^{2d-2} \bmod u(X)$.
   2. Compute $X^q \bmod u(X)$ via square-and-multiply.
   3. Compute and tabulate $X^{qi} \bmod u(X)$ for $i = 2, 3, \ldots, d-1$.
   4. Compute $X^{q^i} \bmod u(X)$ for $i = 2, 3, \ldots, k$, then $X^{q^k} \bmod u(X)$.
2. Do $\gcd\left(v(X)^{(q^k-1)/2} - 1, u(X)\right)$ for random $v(X)$ with $\deg v < \deg u$, to find nontrivial factor $\geq \frac{1}{2}$ of the time; repeat as needed.

To Solve Equation(s) in a Big Tower Field over $\mathbb{F}_q$

Scripted Gröbner Basis Computation

From 3 quadratic equations in 3 variables, we in succession run Gaussian eliminations on matrices of dimensions 3×10, 11×19, 8×16, 5×13, with many coefficients that we know to be zero in advance, to reach a degree-8 equation. You can call this a tailored matrix-$\mathbf{F_4}$.

Cantor-Zassenhaus (instead of Berlekamp)

1. Replace $u(X)$ by $\gcd(u(X), X^{q^k} - X)$ so that u splits in $\mathbb{L}$.
    1. Compute and tabulate $X^d \bmod u(X), \ldots, X^{2d-2} \bmod u(X)$.
    2. Compute $X^q \bmod u(X)$ via square-and-multiply.
    3. Compute and tabulate $X^{qi} \bmod u(X)$ for $i = 2, 3, \ldots, d-1$.
    4. Compute $X^{q^i} \bmod u(X)$ for $i = 2, 3, \ldots, k$, then $X^{q^k} \bmod u(X)$.
2. Toss everything away and repeat unless there is a single solution.

Anything else New For $\mathbb{F}_{2^k}$?

Anything else New For $\mathbb{F}_{2^k}$?

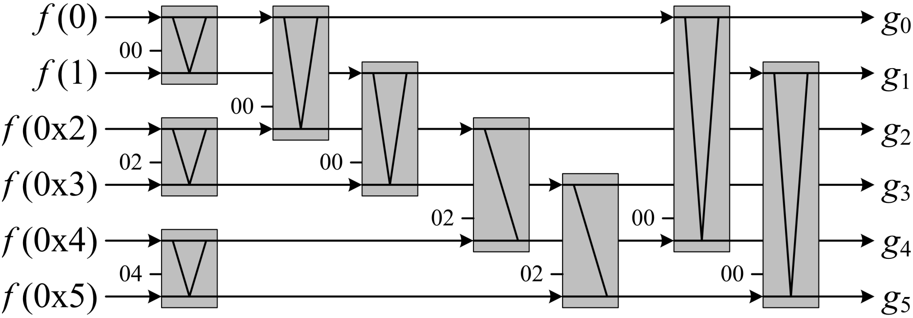
Not Really.

Anything else New For $\mathbb{F}_{2^k}$?

Not Really.

Ok, So we implemented some

- Additive-FFT based multiplication using (V)PSHUFB
- TRUNCATED Additive-FFT too



But no sense talking such with so many sado-masochistic bitslicers here!

Performance on Xeon E3-1245v3 (Haswell) 3.4GHz

Table: 128-bit MPKCs on Intel Haswell.

schemes	gen-key() M cycles	sign() k cycles	verify() k cycles
Rainbow(16,32,32,32)	154.7	89.9	22.8
Rainbow(31,28,28,28)	93.4	77.4	70.8
Rainbow(256,28,20,20)	581.0	121.6	19.0
PFLASH(16,96-1,64)	78.8	226.0	22.6
GUI(2,240,9,16,16,3)	484.2	4,445.4	197.6
GUI(4,120,17,8,8,2)	362.4	11,743.5	1,904.6
HmFEv(256,15,3,16)	201.7	1,497.8	15.7
MQDSS-31-64 [a]	1.827	8,510.6	5,752.6
ECDSA(NIST P256) [b]	0.286	377.1	901.5
Ed25519 [b]	0.066	61.0	185.1
RSA-2048 [b]	233.7	5,240.2	66.4
RSA-3072 [b]	844.4	15,400.9	119.3

[a] on Core i7-4770K (Haswell) 3.5GHz.
[b] eBACS on Xeon E3-1275 v3 (haswell) at 3.5GHz.

Continued: Non-PCLMULQDQ CPUs

We also implemented without PCLMULQDQ, using additive FFT and (V)PSHUFB.

Table: Benchmark of 128-bit MPKCs on SSE-only Platforms

schemes	gen-key() M cycles	sign() k cycles	verify() k cycles
PFLASH(16,96-1,64)	3,269	908.6	32.8
GUI(4,120,17,8,8,2)	510	121,287	1,583.6

Conclusions and Remarks

- It is very important to tune to your architecture.
- MPKCs still competitive speedwise
- Intel's new vector instruction set did double the MPKC throughput.

Thanks for Listening!

- Questions or comments?