SSE Implementation of Multivariate PKCs on Modern X86 CPUs

Chen-Mou Cheng

Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
ccheng@cc.ee.ntu.edu.tw

September 7, 2009

Authors

- This is a joint work with
  - Jintai Ding, University of Cincinnati, USA
  - Bo-Yin Yang, Academia Sinica, Taiwan
  - Students
    - Anna Inn-Tung Chen, University of Michigan, USA
    - Ming-Shing Chen, National Taiwan University, Taiwan
    - Tien-Ren Chen, National Immigration Agency, Taiwan
    - Eric Li-Hsiang Kuo, Academia Sinica, Taiwan
    - Frost Yu-Shuang Lee, University of Michigan, USA

Outline

- Multivariate PKCs
- *SSE*, the x86 vector instruction set extensions
- Using SSSE3 to speed up binary MPKCs
- MPKCs over odd prime fields
- Using SSE2 to speed up odd MPKCs
- Some counter-intuitive (but fast!) techniques
- Performance results

Multivariate PKCs

$$\mathcal{P} : \mathbf{w} \in K^n \overset{S}{\mapsto} \mathbf{x} = \mathbf{M}_S \mathbf{w} + \mathbf{c}_S \overset{\mathcal{Q}}{\mapsto} \mathbf{y} \overset{T}{\mapsto} \mathbf{z} = \mathbf{M}_T \mathbf{y} + \mathbf{c}_T \in K^m$$

- Public map of a typical multivariate PKC over base field $K = \mathbb{F}_q$
  - S and T affine and invertible
  - $\mathcal{Q}$ quadratic, known as as the *central map*
  - For encryption schemes, $n < m$
  - For signature schemes, $n > m$
- Future-proof against quantum computers
- Fast because MPKCs replace arithmetic operations on large units by faster operations on many small units

Unbalanced Oil and Vinegar

$$
M_i := \left[\begin{array}{ccc|ccc}
\alpha_{11}^{(i)} & \cdots & \alpha_{1,v}^{(i)} & \alpha_{1,v+1}^{(i)} & \cdots & \alpha_{1,n}^{(i)} \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
\alpha_{v,1}^{(i)} & \cdots & \alpha_{v,v}^{(i)} & \alpha_{v,v+1}^{(i)} & \cdots & \alpha_{v,n}^{(i)} \\
\hline
\alpha_{v+1,1}^{(i)} & \cdots & \alpha_{v+1,v}^{(i)} & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
\alpha_{n,1}^{(i)} & \cdots & \alpha_{n,v}^{(i)} & 0 & \cdots & 0
\end{array}\right]
$$

Rainbow-like Signatures

- Stage-wise UOV
- For $0 < v_1 < v_2 < \cdots < v_{u+1} = n$
  - $S_l := \{1, 2, \ldots, v_l\}$
  - $O_l := \{v_l + 1, \ldots, v_{l+1}\}$
  - $o_l := v_{l+1} - v_l = |O_l|$
- $\mathcal{Q} : \mathbf{x} = (x_1, \ldots, x_n) \mapsto \mathbf{y} = (y_{v_1+1}, \ldots, y_n)$
  - $y_k := q_k(\mathbf{x})$, with following form if $v_l < k \leq v_{l+1}$

$$q_k = \sum_{i \leq j \leq v_l} \alpha_{ij}^{(k)} x_i x_j + \sum_{i \leq v_l < j < v_{l+1}} \alpha_{ij}^{(k)} x_i x_j + \sum_{i < v_{l+1}} \beta_i^{(k)} x_i$$

- Given all y_i with $v_l < i \leq v_{l+1}$ and all x_j with $j \leq v_l$, we can compute $x_{v_l+1}, \ldots, x_{v_{l+1}}$ via elimination

TTS: Rainbow with Sparse Middle

- Has a sparse $\mathcal{Q}$
- $\mathcal{Q}^{-1}$ needs solving just linear equations, like in Rainbow
- Example from 2004: TTS(20,28)

$$
\begin{aligned}
y_i &= x_i + \textstyle\sum_{j=1}^{7} p_{ij} x_j x_{8+(i+j \bmod 9)},\ i = 8, \ldots, 16 \\
y_{17} &= x_{17} + p_{17,1} x_1 x_6 + p_{17,2} x_2 x_5 + p_{17,3} x_3 x_4 \\
&\quad + p_{17,4} x_9 x_{16} + p_{17,5} x_{10} x_{15} + p_{17,6} x_{11} x_{14} + p_{17,7} x_{12} x_{13} \\
y_{18} &= x_{18} + p_{18,1} x_2 x_7 + p_{18,2} x_3 x_6 + p_{18,3} x_4 x_5 \\
&\quad + p_{18,4} x_{10} x_{17} + p_{18,5} x_{11} x_{16} + p_{18,6} x_{12} x_{15} + p_{18,7} x_{13} x_{14} \\
y_i &= x_i + p_{i,0} x_{i-11} x_{i-9} + \textstyle\sum_{j=19}^{i} p_{i,j-18} x_{2(i-j)-(i \bmod 2)} x_j \\
&\quad + \textstyle\sum_{j=i+1}^{27} p_{i,j-18} x_{i-j+19} x_j,\ i = 19, \ldots, 27
\end{aligned}
$$

The C^* Scheme

- Proposed by Matsumoto and Imai in 1988
- Broken by Patarin in 1995
- The central map is a monomial over $\mathbb{F}_{q^n}$

$$\mathcal{Q}(x) = x^{1+q^\theta} = x \cdot x^{q^\theta}$$

  - $\mathbb{F}_{q^n}$ is an n-dimension vector space over $\mathbb{F}_q$
  - Since $x \mapsto x^q$ is linear, $\mathcal{Q}$ is quadratic
  - Requires that $\gcd(1 + q^\theta, q^n - 1) = 1$
  - $\mathcal{Q}$ is inverted by raising to the inverse power of $1 + q^\theta$

HFE: Hidden Field Equations

- Generalization of C^*
- The central map is a *polynomial* over $\mathbb{F}_{q^n}$

$$\mathcal{Q}(x) = \sum_{q^i+q^j \le D} a_{ij} x^{q^i+q^j} + \sum_{q^i \le D} b_i x^{q^i} + c$$

  - Inversion is much slower than C^*

ℓ-invertible Cycles

- Like C^*, ℓIC also uses an intermediate field $\mathbb{L}^* = \mathbb{K}^k$
- Extends C^* by using the following central map from $(\mathbb{L}^*)^\ell$ to itself

$$\begin{aligned}\mathcal{Q} : (X_1, \ldots, X_\ell) &\mapsto (Y_1, \ldots, Y_\ell) \\ &:= (X_1 X_2, X_2 X_3, \ldots, X_{\ell-1} X_\ell, X_\ell X_1^{q^\alpha})\end{aligned}$$

ℓ-invertible Cycles, $\ell = 3$

- "Standard 3IC," $\ell = 3, \alpha = 0$
- Inversion in $(\mathbb{L}^*)^3$ is easy

$$
\begin{aligned}
\mathcal{Q}: \quad & (X_1, X_2, X_3) \in (\mathbb{L}^*)^3 \mapsto \\
& (X_1 X_2, X_2 X_3, X_3 X_1) \\
\mathcal{Q}^{-1}: \quad & (Y_1, Y_2, Y_3) \in (\mathbb{L}^*)^3 \mapsto \\
& (\sqrt{Y_1 Y_3 / Y_2}, \sqrt{Y_1 Y_2 / Y_3}, \sqrt{Y_2 Y_3 / Y_1},)
\end{aligned}
$$

- Can apply the idea of "intermediate fields" to HFE as well
  - 3HFE, 4HFE, ...
  - Generally faster than HFE

MPKC Modifiers

- All vanilla MPKCs have been broken
- Need modifiers to address attacks
  - Minus (-): throw away some polynomials
  - Prefix or postfix (p): force some $w_i = 0$
- A few others; not used in our implementation

Are MPKCs Still Fast?

- Progress in integer 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
  - AMD Opteron today produces one 128-bit product every 2 cycles
  - Good for ECC!
- 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 number of 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
  - Slower MPKCs but RSA untouched

Questions We Want to Answer

- *Can all the extras on modern commodity CPUs be put to use with MPKCs as well?*
- *If so, how do MPKCs compare to traditional PKCs today, and how is that likely going to change for the future?*

SSE, the X86 Vector Instruction Set Extensions

- SSE: Streaming SIMD Extensions
  - SIMD: Single Instruction Multiple Data
- Most useful: SSE2 integer instructions
  - Work on 16 `xmm` 128-bit registers
  - 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
- SSSE3's PSHUFB also useful

PSHUFB in SSSE3

- 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 $x_{16}, \ldots, x_{31}$ as 0

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

- TT : 16×16 table, with $TT_{i,j} = i * j, 0 \le i, j < 16$
- 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

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

- 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)
  - $\mathtt{xmm} \leftarrow a$-th row of TH
  - $\mathbf{v}' \leftarrow$ PSHUFB $\mathtt{xmm}$,$\mathbf{v}'$
- $\mathbf{v}_i \leftarrow a(\mathbf{v}_i \bmod 16)$
  - $\mathbf{v}_i \leftarrow \mathbf{v}_i \bmod 16$ (AND)
  - $\mathtt{xmm} \leftarrow a$-th row of TL
  - $\mathbf{v} \leftarrow$ PSHUFB $\mathtt{xmm}$,$\mathbf{v}$
- $a\mathbf{v} \leftarrow \mathbf{v} + \mathbf{v}'$ (OR)

Evaluating Public Maps

- Normally we do $z_k = \sum_i w_i \left[P_{ik} + Q_{ik} w_i + \sum_{i<j} R_{ijk} w_j \right]$
- However, the memory access pattern is not good here
- Instead, it is faster if we do
  - $\mathbf{c} \leftarrow [\mathbf{w}^T, (w_i w_j)_{i \le j}]^T$
  - $\mathbf{z} \leftarrow \mathbf{Pc}$, where $\mathbf{P}$ is the $m \times n(n+3)/2$ public-key matrix
  - Due to Faugère and Gilbert

MPKCs over Odd Prime Fields

- Good for defending against Gröbner basis attacks
- The field equation $X^q - X = 0$ becomes much less useful

Basic Building Blocks for Speeding Up Odd MPKCs

- IMULHIb: the upper half in a signed product of two b-bit words
- Useful for computing $\lfloor xy/2^b \rceil$
  - For $-2^{b-1} \le x \le 2^{b-1} - (q-1)/2$
  - $t \leftarrow \text{IMULHI}b\ \lfloor 2^b/q \rceil, x + \lfloor (q-1)/2 \rfloor$
  - $y \leftarrow x - qt$ computes $y = x \bmod q, |y| \le q$
- For $q = 31$ and $b = 16$, we can do even better
  - For $-32768 \le x \le 32752$
  - $t \leftarrow \text{IMULHI16 } 2114, x + 15$
  - $y \leftarrow x - 31t$ computes $y = x \bmod 31, -16 \le y \le 15$

Remarks on Getting More Performance

- Laziness often leads to optimality
  - Do not always need the tightest range
  - The less reductions, 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

Speeding Up Polynomial Evaluation

- PMADDWD: Packed Multiply and Add, Word to Double-word
  - Source: $(x_0, \ldots, x_7)$
  - Destination: $(y_0, \ldots, y_7)$
  - Result: $(x_0 y_0 + x_1 y_1, x_2 y_2 + x_3 y_3, x_4 y_4 + x_5 y_5, x_6 y_6 + x_7 y_7)$
- Helpful in evaluating $\mathbf{z} = \mathbf{Pc}$, piece by piece
  - Let $\mathbf{Q}$ be a 4×2 submatrix of $\mathbf{P}$
  - $\mathbf{d}^T$ be the corresponding 2×1 submatrix of $\mathbf{c}$
  - r1 $\leftarrow (Q_{11}, Q_{12}, Q_{21}, Q_{22}, Q_{31}, Q_{32}, Q_{41}, Q_{42})$
  - r2 $\leftarrow (d_1, d_2, d_1, d_2, d_1, d_2, d_1, d_2)$
  - PMADDWD r1, r2 computes $\mathbf{Qd}$
  - Continue in 32-bits until reduction $\bmod q$
- Saves a few $\bmod q$ operations and delivers $1.5\times$ performance

Inversion in $\mathbb{F}_{31}$

- Normally do table look-ups
- Alternative: $x \mapsto x^{29}$
  - $y \leftarrow x * x * x \bmod 31$ $(y = x^3)$
  - $y \leftarrow x * y * y \bmod 31$ $(y = x^7)$
  - $y \leftarrow y * y \bmod 31$ $(y = x^{14})$
  - $y \leftarrow x * y * y \bmod 31$ $(y = x^{29})$
- Deliver $2\times$ performance over table look-ups!

Wiedemann vs. Gauss Elimination

- How to solve a medium-sized dense linear system?
  - Wiedemann iterative solver for $\mathbf{A}\mathbf{x} = \mathbf{b}$
    - Compute $\mathbf{z}\mathbf{A}^i\mathbf{b}$ 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
- Result: Wiedemann beats Gauss by a factor of 2!

Special Tower Fields

- $\mathbb{F}_{q^k}$ isomorphic to $\mathbb{F}_q\,[t]/p(t)$, $\deg p = k$ and p irreducible
- For $k|(q-1)$ and a few other cases, $p(t) = t^k - a$ for a small a
  - Deliver $2\times$ reduction performance over cases where p has 3 terms
  - $X \mapsto X^q$ becomes very easy to compute
  - Multiplication and division are also very easy
  - Inversion: (again) raising to the $(q^k - 2)$-th power!
- Square roots computed via Tonelli-Shanks
- Univariate equations solved via Cantor-Zassenhaus

Performance Comparison on Intel Q9550

Scheme	Result	PubKey	PriKey	KeyGen	PubMap	PriMap
RSA (1024 bits)	128 B	128 B	1024 B	27.2 ms	26.9 μs	806.1 μs
4HFE-p (31,10)	68 B	23 KB	8 KB	4.1 ms	6.8 μs	659.7 μs
3HFE-p (31,9)	67 B	7 KB	5 KB	0.8 ms	2.3 μs	60.5 μs
RSA (1024 bits)	128 B	128 B	1024 B	26.4 ms	22.4 μs	813.5 μs
ECDSA (160 bits)	40 B	40 B	60 B	0.3 ms	409.2 μs	357.8 μs
C^*-p (pFLASH)	37 B	72 KB	5 KB	28.7 ms	97.9 μs	473.6 μs
3IC-p (31,18,1)	36 B	35 KB	12 KB	4.2 ms	11.7 μs	256.2 μs
Rainbow (31,24,20,20)	43 B	57 KB	150 KB	120.4 ms	17.7 μs	70.6 μs
TTS (31,24,20,20)	43 B	57 KB	16 KB	13.7 ms	18.4 μs	14.2 μs

Measured using SUPERCOP: System for unified performance evaluation related to cryptographic operations and primitives. `http://bench.cr.yp.to/supercop.html`, April 2009.

Concluding Remarks

- Take-away point: Odd MPKCs worth studying!
  - Algebraic attacks become harder
  - Friendly to mainstream computing devices
    - X86 CPUs have vector instructions
    - High-end FPGAs have multiplier IPs
    - Also good for many-core GPUs (NVIDIA, ATI/AMD, Larrabee)

Thanks for Listening!

- Questions or comments?