Elliptic Curve Cryptography

聚安全--移动安全沙龙(上海)

李枫

2014.12.20

Content

I. Background

- Fermat's Last Theorem
- Field & Group
- Definitions
- Domain Parameters
- ECC vs RSA
- IT Standards

II. SW Implementation

- Overview
- My Reference Implementation
- Java
- Python
- C++/C
- ARM

Content

III. HW Implementation

- Instruction Set Extension
- Crypto Coprocessor/Accelerator
- GPGPU
- FPGA

IV. Application

- Bitcoin
- e-IDMS (Europe)

V. Reference

I. Background

1) Fermat's Last Theorem

- http://en.wikipedia.org/wiki/Fermat%27s_Last_Theorem

Diophantine equation $x^n + y^n = z^n$ has no integer solutions for $n > 2$ and $x, y, z \neq 0$.

- In 1984, Gerhard Frey noted a link between Fermat's equation and the modularity theorem, then still a conjecture. If Fermat's equation had any solution (a, b, c) for exponent p > 2, then it could be shown that the elliptic curve would have such unusual properties that it was unlikely to be modular

$$y^2 = x(x - a^p)(x + b^p)$$

- In 1993, **Andrew John Wiles** presented his proof to the public for the first time at a conference in Cambridge
- In August 1993 it was discovered that the proof contained a flaw in one area
- Together with his former student **Richard Taylor,** he published a second paper which circumvented the problem and thus completed the proof in **1995**
- http://www.bbc.co.uk/programmes/b0074rxx

Sir Andrew John Wiles

2) Field & Group

Finite Field

- **http://en.wikipedia.org/wiki/Finite_field**

Fields are abstractions of familiar number systems (such as the rational numbers $\mathbb{Q}$, the real numbers $\mathbb{R}$, and the complex numbers $\mathbb{C}$) and their essential properties. They consist of a set $\mathbb{F}$ together with two operations, addition (denoted by $+$) and multiplication (denoted by $\cdot$), that satisfy the usual arithmetic properties:

(i) $(\mathbb{F}, +)$ is an abelian group with (additive) identity denoted by 0.

(ii) $(\mathbb{F} \setminus \{0\}, \cdot)$ is an abelian group with (multiplicative) identity denoted by 1.

(iii) The distributive law holds: $(a+b) \cdot c = a \cdot c + b \cdot c$ for all $a, b, c \in \mathbb{F}$.

If the set $\mathbb{F}$ is finite, then the field is said to be *finite*.

Abelian Group

- **http://en.wikipedia.org/wiki/Abelian_group**

An *abelian group* $(G, *)$ consists of a set G with a binary operation $* : G \times G \to G$ satisfying the following properties:

(i) (*Associativity*) $a * (b * c) = (a * b) * c$ for all $a, b, c \in G$.

(ii) (*Existence of an identity*) There exists an element $e \in G$ such that $a * e = e * a = a$ for all $a \in G$.

(iii) (*Existence of inverses*) For each $a \in G$, there exists an element $b \in G$, called the *inverse* of a, such that $a * b = b * a = e$.

(iv) (*Commutativity*) $a * b = b * a$ for all $a, b \in G$.

3) Definitions

Weierstrass equation

- An elliptic curve **E** over a field is defined by an equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

— For the prime finite fields $GF(p)$ with $p > 3$, the Weierstrass equation is

$$y^2 = x^3 + ax + b$$

where a and b are integers modulo p for which $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$.

— For the binary finite fields $GF(2^m)$, the Weierstrass equation is

$$y^2 + xy = x^3 + ax^2 + b$$

where a and b are elements of $GF(2^m)$ with $b \neq 0$.

- **Point at infinity & the order of E**

Given a Weierstrass equation, the elliptic curve E consists of the solutions (x, y) over $GF(q)$ to the defining equation, along with an additional element called the *point at infinity* (denoted O). The points other than O are called *finite* points. The number of points on E (including O) is called the *order* of E and is denoted by $\#E(GF(q))$.

The point at infinity O plays a role analogous to that of the number 0 in ordinary addition. Thus

$$P + O = P$$

$$P + (-P) = O$$

for all points P.

Example

Example: Let E be the curve

$$y^2 = x^3 + 10x + 5$$

over the field $GF(13)$. Then the points on E are

$$\{O, (1,4), (1,9), (3,6), (3,7), (8,5), (8,8), (10,0), (11,4), (11,9)\}$$

Thus, the order of E is $\#E(GF(13)) = 10$.

Example: Let E be the curve

$$y^2 + xy = x^3 + (t + 1)\, x^2 + 1$$

over the field $GF(2^3)$ given by the polynomial basis with field polynomial $t^3 + t + 1 = 0$. Then the points on E are

$$\begin{aligned}&\{O, ((000), (001))\\ &((010), (100)), ((010), (110)), ((011), (100)), ((011), (111)),\\ &((100), (001)), ((100), (101)), ((101), (010)), ((101), (111)),\\ &((110), (000)), ((110), (110)), ((111), (001)), ((111), (110))\}\end{aligned}$$

Thus, the order of E is $\#E(GF(2^3)) = 14$.

This representation is determined by choosing an irreducible binary polynomial $p(t)$ of degree m. (See A.3 and A.4 for definitions of the above terms and for a description of the arithmetic of a field using this representation.) If the polynomial basis representation over $GF(2)$ is used, then, for purposes of conversion, the bit string

$$(a_{m-1} \ldots a_2\, a_1\, a_0)$$

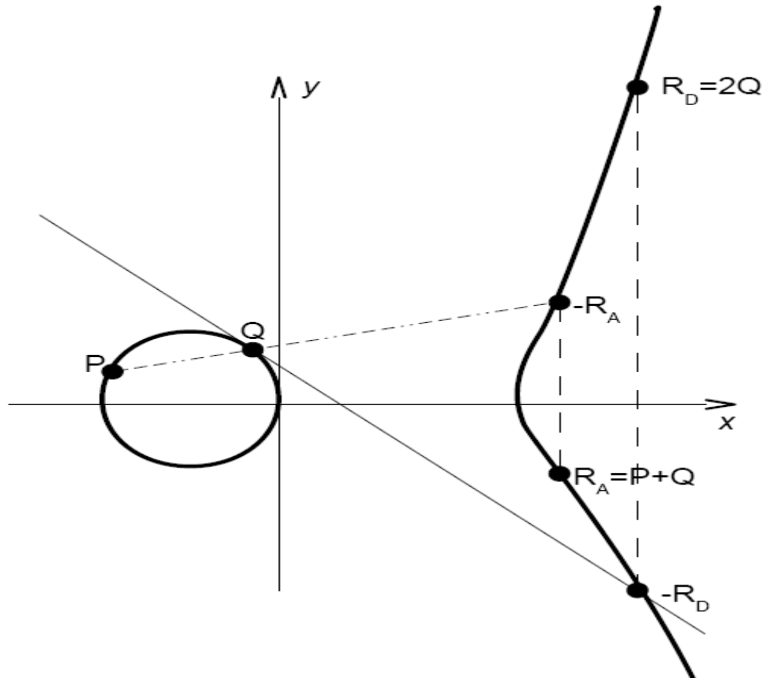
shall be taken to represent the polynomial

$$a_{m-1}t^{m-1} + \ldots + a_2t^2 + a_1t + a_0$$

where the coefficients a_i are elements of $GF(2)$.

Full addition

- **prime field**



General form of Weierstrass' curves on GF(p):

$$y^2 \bmod p = (x^3 + ax + b) \bmod p$$

Define the *inverse* of the point $P = (x, y)$ to be

$$-P = \begin{cases} (x, -y) \text{ if } q = p \text{ prime}, \\ (x, x+y) \text{ if } q = 2^m \end{cases}$$

Inversion

Recall that the inverse of a nonzero element $a \in \mathbb{F}_p$, denoted $a^{-1} \bmod p$ or simply a^{-1} if the field is understood from context, is the unique element $x \in \mathbb{F}_p$ such that $ax = 1$ in $\mathbb{F}_p$, i.e., $ax \equiv 1 \pmod p$. Inverses can be efficiently computed by the extended Euclidean algorithm for integers.

Rule to add two points with different x-coordinates: Let $(x_1, y_1) \in E(\mathbb{F}_p)$ and $(x_2, y_2) \in E(\mathbb{F}_p)$ be two points such that $x_1 \neq x_2$. Then $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, where:

$$x_3 \equiv \lambda^2 - x_1 - x_2 \pmod p, \quad y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod p, \quad \text{and } \lambda \equiv \frac{y_2 - y_1}{x_2 - x_1} \pmod p.$$

Rule to add a point to itself (double a point): Let $(x_1, y_1) \in E(\mathbb{F}_p)$ be a point with $y_1 \neq 0$. Then $(x_1, y_1) + (x_1, y_1) = (x_3, y_3)$, where:

$$x_3 \equiv \lambda^2 - 2x_1 \pmod p, \quad y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod p, \quad \text{and } \lambda \equiv \frac{3x_1^2 + a}{2y_1} \pmod p.$$

Scalar multiplication

- Elliptic curve points can be added but not *multiplied*. It is, however, possible to perform *scalar multiplication*, which is another name for repeated addition of the same point. If n is a positive integer and P a point on an elliptic curve, the scalar multiple nP is the result of adding n copies of P. Thus, for example, $5P = P + P + P + P + P$.

  The notion of scalar multiplication can be extended to zero and the negative integers via

  $$0P = \mathcal{O}, \qquad (-n)\,P = n\,(-P)$$

Point Order

- The *order* of a point P on an elliptic curve is the smallest positive integer r such that $rP = \mathcal{O}$. The order always exists and divides the order of the curve $\#E(GF\,(q))$. If k and l are integers, then $kP = lP$ if, and only if, $k \equiv l \pmod r$.

Elliptic curve discrete logarithms

- Suppose that the point G on E has prime order r, where r^2 does not divide the order of the curve $\#E(GF\,(q))$. Then, a point P satisfies $P = lG$ for some l if, and only if, $rP = \mathcal{O}$. The coefficient l is called the *elliptic curve discrete logarithm* of P (with respect to the base point G). The elliptic curve discrete logarithm is an integer modulo r.

Projective coordinates

- If division within $GF\,(q)$ is relatively expensive, then it may pay to keep track of numerators and denominators separately. In this way, one can replace division by α with multiplication of the denominator by α. This is accomplished by the projective coordinates X, Y, and Z given by

  $$x = \frac{X}{Z^2},\quad y = \frac{Y}{Z^3}$$

  $$Y^2 = X^3 + aXZ^4 + bZ^6$$

4) Domain Parameters
prime field

$\boxed{q}$	The size of the underlying field used (part of the EC domain parameters)
$\boxed{a, b}$	The coefficients defining the elliptic curve E, elements of $GF(q)$ (part of the EC domain parameters)
E	The elliptic curve over the field $GF(q)$ defined by a and b
$\#E$	The number of points on the elliptic curve E
$\boxed{r}$	The prime divisor of $\#E$ and the order of G (part of the EC domain parameters)
$\boxed{G}$	A curve point generating a subgroup of order r (part of the EC domain parameters)
k	$\#E/r$, the cofactor
s, u, s', u'	EC private keys, integers, corresponding to public keys W, V, W', V', respectively
W, V, W', V'	EC public keys, points on the curve, corresponding to private keys s, u, s', u', respectively
$(s, W), (u, V)$	EC key pairs, where s and u are the private keys, and W and V are the corresponding public keys

5) ECC vs RSA

	RSA	ECC
Discovery	1977 (previously discovered in 1969 by GHCQ and perhaps earlier by NSA)	1985 (adoption limited until ~2005)
"Hard" Problem	Factoring	Discrete Log on Elliptic Curve
Key Size (~112-bit)	**2048 bits** (768 bits broken)	**224 bits** (112 bits broken)
Backdoor Risk	None	Curves selected by NSA
Quantum Computing Risk	Known fast factoring algorithms (Shor's)	Similar (variation of Shor's algorithm solves Discrete Log)
Implementation Challenges	Avoiding weak keys, timing side channels	Fast operations on elliptic curves, leaks on invalid inputs

Key-size Equivalence

Security (bits)	RSA	DLOG		EC
		field size	subfield	
48	480	480	96	96
56	640	640	112	112
64	816	816	128	128
80	1248	1248	160	160
112	2432	2432	224	224
128	3248	3248	256	256
160	5312	5312	320	320
192	7936	7936	384	384
256	15424	15424	512	512

6) IT Standards

IEEE P1363

- **http://grouper.ieee.org/groups/1363/ (1363-2000 & 1363a-2004)**
- **http://en.wikipedia.org/wiki/IEEE_P1363**

– *Primitives:* Basic mathematical operations. Historically, they were discovered based on number-theoretic hard problems. Primitives are not meant to achieve security by themselves, but they serve as building blocks for schemes.

– *Schemes:* A collection of related operations combining primitives and additional methods (see 4.4). Schemes can provide complexity-theoretic security, which is enhanced when they are appropriately applied in protocols.

– *Protocols:* Sequences of operations to be performed by multiple parties to achieve some security goal. Protocols can achieve desired security for applications if implemented correctly.

NIST FIPS 186

- **http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf**
  (Digital Signature Standard, Issued July 2013)

ANSI X9.62

- **http://webstore.ansi.org/RecordDetail.aspxsku=ANSI+X9.62%3a2005**
  Public Key Cryptography for the Financial Services Industry,
  The Elliptic Curve Digital Signature Algorithm (ECDSA)

SEC

- **secg.org**
- **SEC 1: Elliptic Curve Cryptography (v2.0, issued May, 2009)**
- **SEC 2: Recommended Elliptic Curve Domain Parameters) (v2.0, issued Jan, 2010)**

certicom

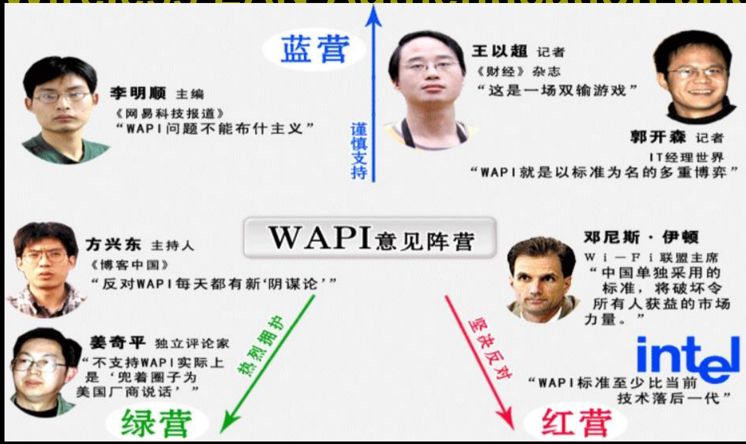
BlackBerry

President Barack Obama displaying his Blackberry to reporters after nearly leaving it behind ahead of a flight to Vegas on November 21, 2014.

WAPI

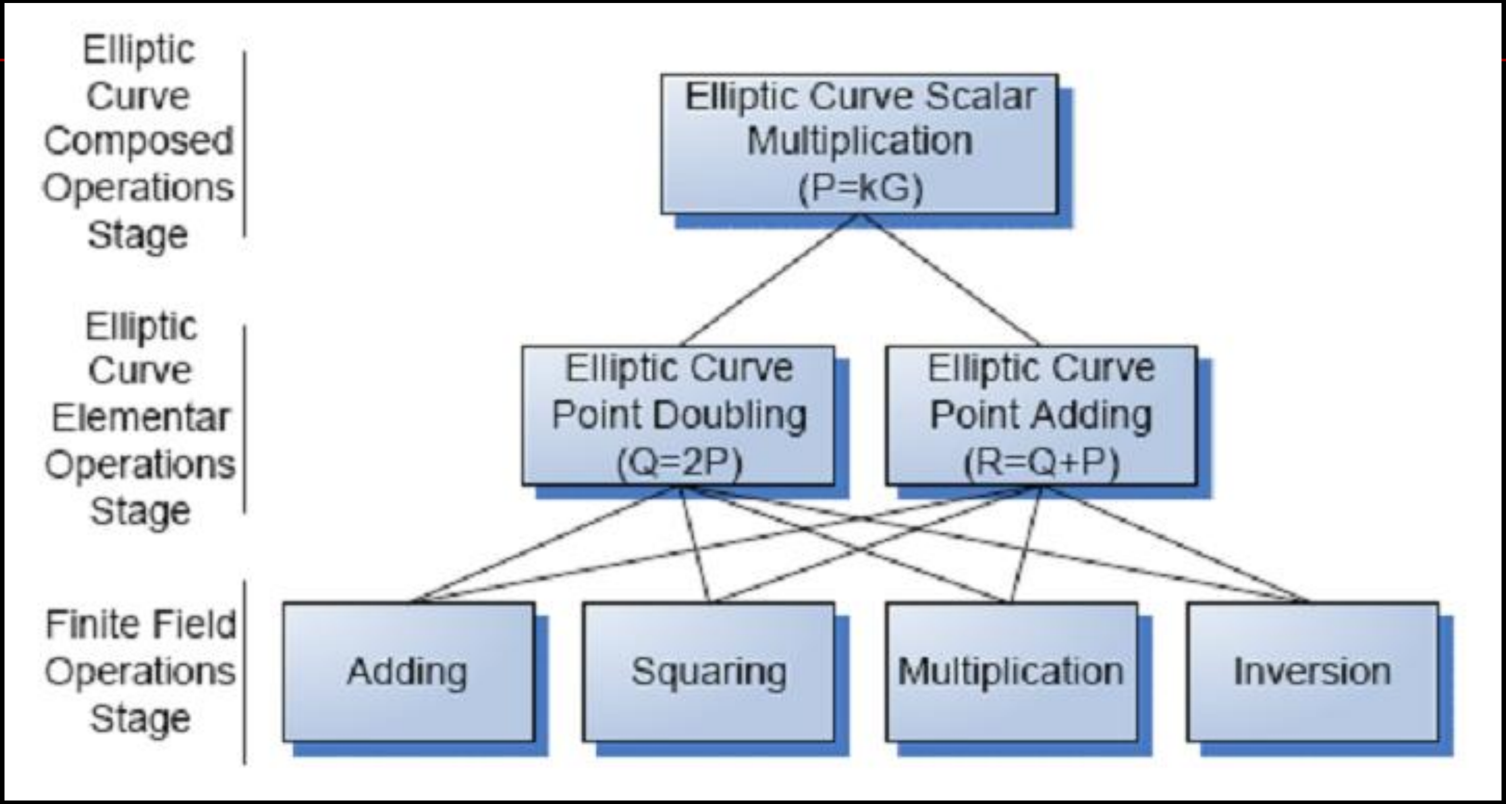
- **www.sac.gov.cn**

Wireless LAN Authentication and Privacy Infrastructure



II. SW Implementation

1) Overview



For example, let kP be a scalar multiplication, where $k = 27$ (binary 11011) and P is a point on the elliptic curve E. The steps performed by the left-to-right Double-and-Add method for computing $27P$ are:

$$2 \times (2 \times (2 \times (2 \times ((2 \times O) + P) + P)) + P) + P$$

A.10.3 Elliptic scalar multiplication

Scalar multiplication can be performed efficiently by the *addition-subtraction method* outlined below.

Input: An integer n and an elliptic curve point P

Output: The elliptic curve point nP

1. If $n = 0$, then output O and stop.
2. If $n < 0$, then set $Q \leftarrow (-P)$ and $k \leftarrow (-n)$; else set $Q \leftarrow P$ and $k \leftarrow n$.
3. Let $h_l\, h_{l-1} \ldots h_1\, h_0$ be the binary representation of $3k$, where the most significant bit h_l is 1.
4. Let $k_l\, k_{l-1} \ldots k_1\, k_0$ be the binary representation of k.
5. Set $S \leftarrow Q$.
6. For i from $l - 1$ downto 1 do

   Set $S \leftarrow 2S$.

   If $h_i = 1$ and $k_i = 0$, then compute $S \leftarrow S + Q$ via A.10.1 or A.10.2.

   If $h_i = 0$ and $k_i = 1$, then compute $S \leftarrow S - Q$ via A.10.1 or A.10.2.
7. Output S.

A.10.4 Projective elliptic doubling (prime case)

Input: A modulus p; the coefficients a and b defining a curve E modulo p; projective coordinates (X_1, Y_1, Z_1) for a point P_1 on E

Output: Projective coordinates (X_2, Y_2, Z_2) for the point $P_2 = 2P_1$

1. $T_1 \leftarrow X_1$.
2. $T_2 \leftarrow Y_1$.
3. $T_3 \leftarrow Z_1$.
4. If $T_2 = 0$ or $T_3 = 0$, then output $(1, 1, 0)$ and stop.
5. If $a = p - 3$ then

   $T_4 \leftarrow T_3^2$

   $T_5 \leftarrow T_1 - T_4$

   $T_4 \leftarrow T_1 + T_4$

   $T_5 \leftarrow T_4 \times T_5$

   $T_4 \leftarrow 3 \times T_5$ (this step computes M)

   else

   $T_4 \leftarrow a$

   $T_5 \leftarrow T_3^2$

   $T_5 \leftarrow T_5^2$

   $T_5 \leftarrow T_4 \times T_5$

   $T_4 \leftarrow T_1^2$

   $T_4 \leftarrow 3 \times T_4$

   $T_4 \leftarrow T_4 + T_5$ (this step computes M).
6. $T_3 \leftarrow T_2 \times T_3$.
7. $T_3 \leftarrow 2 \times T_3$ (this step computes Z_2).
8. $T_2 \leftarrow T_2^2$.
9. $T_5 \leftarrow T_1 \times T_2$.
10. $T_5 \leftarrow 4 \times T_5$ (this step computes S).
11. $T_1 \leftarrow T_4^2$.
12. $T_1 \leftarrow T_1 - 2 \times T_5$ (this step computes X_2).
13. $T_2 \leftarrow T_2^2$.
14. $T_2 \leftarrow 8 \times T_2$ (this step computes T).
15. $T_5 \leftarrow T_5 - T_1$.
16. $T_5 \leftarrow T_4 \times T_5$.
17. $T_2 \leftarrow T_5 - T_2$ (this step computes Y_2).
18. $X_2 \leftarrow T_1$.
19. $Y_2 \leftarrow T_2$.
20. $Z_2 \leftarrow T_3$.

- **Approximated number of point additions and doubling of scalar multiplication, for different scalar representations**

Representation	Doubling	Additions
Binary	$n-1$	$(n-1)/2$
NAF	n	$n/3$
w-NAF	$n+1$	$n/(w+1)$
MOF	n	$n/3$
w-MOF	$n+1$	$n/(w+1)$

- **Operations required for point addition and doubling on GF(p). M = Multiplication, S = Square and I = Inversion, Affine coordinate (A), projective coordinates: Standard (P), Jacobian (J)…**

Doubling		Addition	
Operation	Cost	Operation	Cost
$2\mathcal{P}$	$7M+5S$	$\mathcal{P}+\mathcal{P}$	$12M+2S$
$2\mathcal{J}^c$	$5M+6S$	$\mathcal{J}^c+\mathcal{J}^c$	$11M+3S$
$2\mathcal{J}$	$4M+6S$	$\mathcal{J}+\mathcal{J}$	$12M+4S$
$2\mathcal{J}^m$	$4M+4S$	$\mathcal{J}^m+\mathcal{J}^m$	$13M+6S$
$2\mathcal{A}$	$I+2M+2S$	$\mathcal{A}+\mathcal{A}$	$I+2M+S$

Algorithm Agility: the benefits

ECC

1 Stronger Encryption

- Shorter key than RSA
- 256-bit ECC = 3072-bit RSA
- 10k times harder to crack than RSA 2048
- Meets NIST recommendations

2 Efficient Performance

- Efficiency increases with higher server loads
- Utilises less server CPU
- PC's: Faster page load time
- Ideal for mobile devices

3 Highly Scalable

- Large SSL deployments w/out additional hardware
- Securing the enterprise:
  - Use fewer resources
  - Lower costs

4 Future of Crypto Tech

- Viable for many years
- Built for Internet of things to come
- Supports billions of new devices coming online
- Ideal for Open Networks
- Truly 'future proof" trust infrastructure in place.

Binary field multiplication in SW

- **shift-and-add era**

Year	Technology	Work	Field	Cycles
1996	Intel Pentium 133 MHz (32-bit)	**A Fast Software Implementation for Arithmetic Operations in $GF(2^n)$** *De Win, Bosselaers, Vandenberghe, De Gersem and Vandewalle* ASIACRYPT	$\mathbb{F}_{2^{176}}$ $\mathbb{F}_{(2^{16})^{11}}$	8,445

- **comb (w = 4) era**

Year	Technology	Work	Field	Cycles
2004	Intel Pentium III 800 MHz (32-bit, MMX)	**Field Inversion and Point Halving Revisited** *Fong, Hankerson, López and Menezes* IEEE Transactions on Computers	$\mathbb{F}_{2^{163}}$ $\mathbb{F}_{2^{233}}$	560 1,840

- **Karatsuba**

Year	Technology	Work	Field	Cycles
2013	Intel Xeon 3.4 GHz "Haswell" (64-bit, SSE4.2, AVX2, PCLMULQDQ[7cc])	**Personal Benchmarking** *Oliveira, López, Aranha and Rodríguez-Henríquez*	$\mathbb{F}_{2^{254}}$ $\mathbb{F}_{(2^{127})^2}$	44

2) My Reference Implementation (via Java)

Big Integer

java.math

Class BigInteger

java.lang.Object
  java.lang.Number
    java.math.BigInteger

All Implemented Interfaces:

Serializable, Comparable<BigInteger>

```java
/**
 * The magnitude of this BigInteger, in <i>big-endian</i> order: the
 * zeroth element of this array is the most-significant int of the
 * magnitude.  The magnitude must be "minimal" in that the most-significant
 * int ({@code mag[0]}) must be non-zero.  This is necessary to
 * ensure that there is exactly one representation for each BigInteger
 * value.  Note that this implies that the BigInteger zero has a
 * zero-length mag array.
 */
final int[] mag;
```

Operations on binary field

java.util

Class BitSet

java.lang.Object
  java.util.BitSet

All Implemented Interfaces:

Serializable, Cloneable

```java
/**
 * The internal field corresponding to the serialField "bits".
 */
private long[] words;

/**
 * The number of words in the logical size of this BitSet.
 */
private transient int wordsInUse = 0;
```

BitSet32

prime case

Coordinates	Jacobian Projective coordinates
Base class for field arithmetic	java.math.BigInteger
Scalar multiplication	IEEE P1363 std-2000 section A.10.9
Projective full addition and subtraction	IEEE P1363 std-2000 section A.10.8
projective elliptic addition	IEEE P1363 std-2000 section A.10.5
Projective elliptic doubling	IEEE P1363 std-2000 section A.10.4
#E	Naive-Attempt algorithm
Coordinate of the point at infinity	（1，1，0）（Recommanded by IEEE P1363）

binary case

Coordinates	Affine coordinates
Basis	Normal basis
Base class for field arithmetic	Bitset32
Polynomial multiplication	Right-to-left comb algorithm
Polynomial squaring	Replacement algorithm
Inversion and division	Shantz algorithm
Reduction	SOOS algorithm
Scalar multiplication	IEEE P1363 std-2000 section A.10.3
Full addition and subtraction	IEEE P1363 std-2000 section A.10.2
Elliptic addition	refer to IEEE P1363 std-2000 section A.10.5
Elliptic doubling	refer to IEEE P1363 std-2000 section A.10.4
#E	N/A
Coordinate of the point at infinity	（0，0）（Recommanded by IEEE P1363）

3) Java

IAIK-JCE

- http://jce.iaik.tugraz.at/

JCA/JCE

The IAIK Provider for the Java™ Cryptography Extension (IAIK-JCE) is a set of APIs and implementations of cryptographic functionality, including hash functions, message authentication codes, symmetric, asymmetric, stream, and block encryption, key and certificate management. It supplements the security functionality of the default JDK.

Download the Product Highlights Brochure!

The architecture of the IAIK-JCE follows the same design principles as found elsewhere in the JCA.

Features

- Contains re-implementation of the whole Java™ Cryptography Extension (JCE) framework
- Extensive Security Provider
- Built-in ASN.1 library
- Support for many PKCS standards
- X.509 certificate and CRL handling for building PKI solutions
- Ldap Certificate/Crl Search utilities
- Secure Random number generators
- Special versions for applets and Java™ WebStart

IAIK JCE SE Basic License

IAIK JCE Basic License (non-US version): Single developer license	first license € 600 additional: € 150
IAIK JCE Basic License (non-US-version): Unlimited developer license	€ 9.000
IAIK JCE Basic License (US version):Single developer license	first license € 600 additional: € 150
IAIK JCE Basic License (US-version): Unlimited developer license	€ 9.000

JCE CC

Common Criteria EAL 3+ and EAL 3 evaluated IAIK-JCE CC Core

IAIK-JCE CC Core 3.15 and 3.1 (incl. JCE unlimited developer license) **Note:** This product will always be delivered on CD!	€ 15.000

Bouncy Castle

- http://bouncycastle.org/

```
import org.bouncycastle.jce.spec.ECParameterSpec;
...
ECParameterSpec ecSpec = ECNamedCurveTable.getParameterSpec("prime192v1");
KeyPairGenerator g = KeyPairGenerator.getInstance("ECDSA", "BC");
g.initialize(ecSpec, new SecureRandom());
KeyPair pair = g.generateKeyPair();
```

JDK 1.5 - JDK 1.7
JDK 1.4
JDK 1.3
JDK 1.2
JDK 1.1

4) Python

cryptography

- https://cryptography.io/en/latest/

Why a new crypto library for Python?

If you've done cryptographic work in Python before, you've probably seen some other libraries in Python, such as *M2Crypto*, *PyCrypto*, or *PyOpenSSL*. In building `cryptography` we wanted to address a few issues we observed in the existing libraries:

- Lack of PyPy and Python 3 support.
- Lack of maintenance.
- Use of poor implementations of algorithms (i.e. ones with known side-channel attacks).
- Lack of high level, "Cryptography for humans", APIs.
- Absence of algorithms such as `AES-GCM` and `HKDF`.
- Poor introspectability, and thus poor testability.
- Extremely error prone APIs, and bad defaults.

Sage

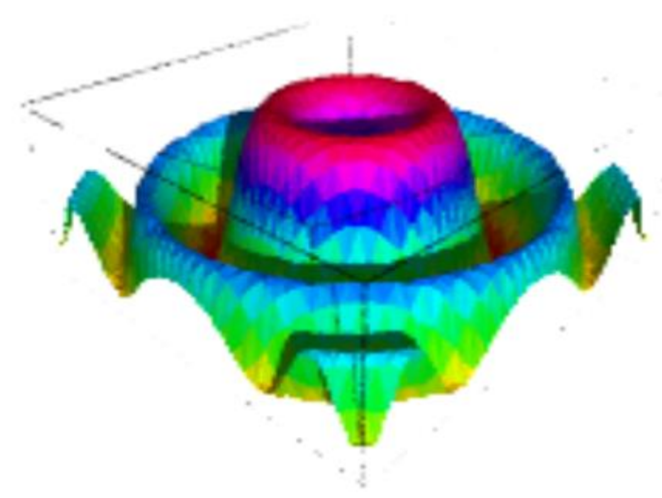
- http://sagemath.org/

Sage is a free open-source mathematics software system licensed under the GPL. It builds on top of many existing open-source packages: NumPy, SciPy, matplotlib, Sympy, Maxima, GAP, FLINT, R and many more.

We now give a more interesting case, the NIST-P521 curve. Its order is too big to calculate with SAGE, and takes a long time using other packages, so it is very useful here.

```
sage: p = 2^521 - 1
sage: prev_proof_state = proof.arithmetic()
sage: proof.arithmetic(False) # turn off primality checking
sage: F = GF(p)
sage: A = p - 3
sage: B = 1093849038073734274511112390766805569936207598951683748994586394495953116150735016
sage: q = 6864797660130609714981900799081393217269435300143305409394463459185543183397655394
sage: E = EllipticCurve([F(A), F(B)])
sage: G = E.random_point()
sage: G.set_order(q)
sage: G.order() * G  # This takes practically no time.
(0 : 1 : 0)
sage: proof.arithmetic(prev_proof_state) # restore state
```

$$f = \frac{sin(y*y+x*x)}{\sqrt{(x*x+y*y+.0001)}} : plot3d(f,(-3,3),(-3,3))$$



Large contributors community:

5) C++/C

Crypto++

- http://en.wikipedia.org/wiki/Crypto%2B%2B
- http://www.cryptopp.com/

Crypto++ algorithms and implementations

Primitive or Operation	Algorithms or Implementations
Pseudorandom number generators	LCG, KDF2, Blum Blum Shub, ANSI X9.17
High speed stream ciphers	Panama, SOSEMANUK, Salsa20, XSalsa20
AES and AES candidates	Rijndael (AES selection), RC6, MARS, Twofish, Serpent, CAST-256
Other block ciphers	IDEA, Triple-DES (DES-EDE2 and DES-EDE3), Camellia, SEED, RC5, Blowfish, TEA, XTEA, Skipjack, SHACAL-2
Block cipher modes of operation	ECB, CBC, CTS, CFB, OFB, CTR
Authenticated encryption modes	CCM, GCM, EAX
Block ciphers padding schemes	PKCS#5, PKCS#7, Zeros, One and zeros
Message authentication codes	VMAC, HMAC, CMAC, CBC-MAC, DMAC, Two-Track-MAC
Cryptographic hash function	SHA-1, SHA-2 (SHA-224, SHA-256, SHA-384, and SHA-512), SHA-3, Tiger, WHIRLPOOL, RIPEMD (RIPEMD-128, RIPEMD-160, RIPEMD-256, and RIPEMD-320)
Password based key derivation functions	PBKDF1 and PBKDF2 from PKCS #5, PBKDF from PKCS #12 appendix B
Public-key cryptography	RSA, DSA, ElGamal, Nyberg-Rueppel (NR), Rabin-Williams (RW), LUC, LUCELG, DLIES (variants of DHAES), ESIGN
Padding schemes for public-key systems	PKCS#1 v2.0, OAEP, PSS, PSSR, IEEE P1363 EMSA2 and EMSA5
Key agreement schemes	Diffie-Hellman (DH), Unified Diffie-Hellman (DH2), Menezes-Qu-Vanstone (MQV), LUCDIF, XTR-DH
Elliptic curve cryptography	ECDSA, ECNR, ECIES, ECDH, ECMQV
Secret Sharing	Shamir's secret sharing scheme, Rabin's information dispersal algorithm (IDA)

LibTom

- http://www.libtom.org/
- https://github.com/libtom/libtomcrypt
- https://github.com/libtom/libtommath

Botan

- **http://botan.randombit.net/**

> **Note**
>
> Versions 1.11.0 and later require a mostly-compliant C++11 compiler such as Clang 3.1 or GCC 4.7.

State of native implementations

- 

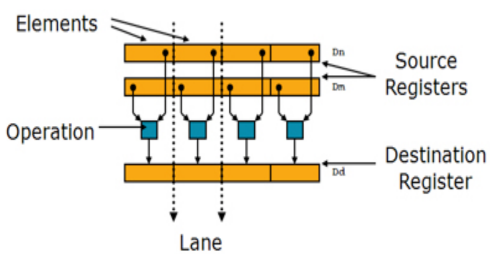
6) ARM

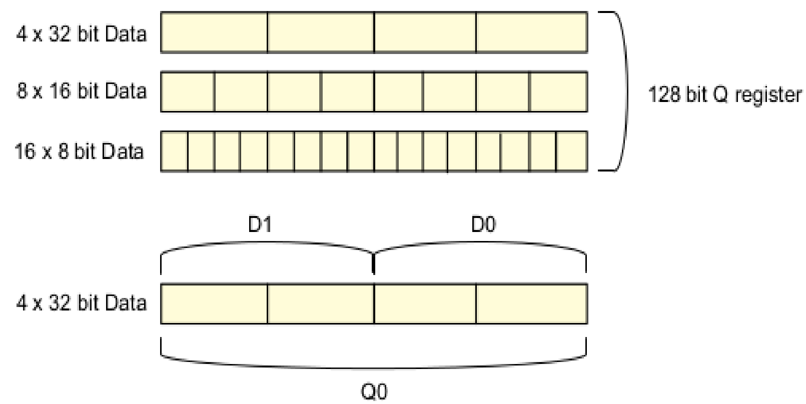
NEON

- http://www.arm.com/products/processors/technologies/neon.php

NEON instructions perform "Packed SIMD" processing:

- Registers are considered as **vectors** of **elements** of the same **data type**
- Data types can be: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, single precision floating point
- Instructions perform the same **operation** in all **lanes**





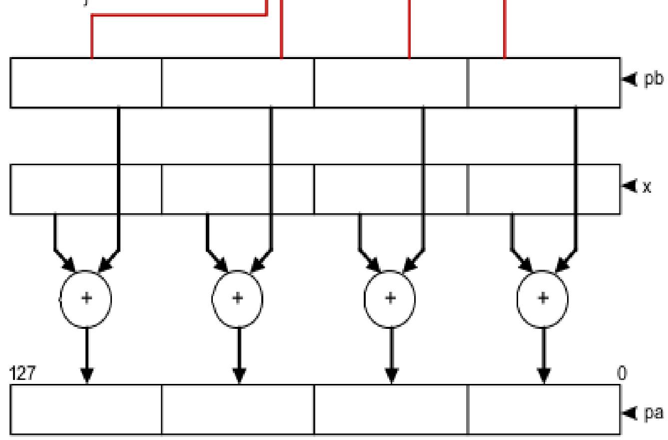
In the following example, array A contains eight 16-bit elements. The example shows how to load this data from this array into a vector.

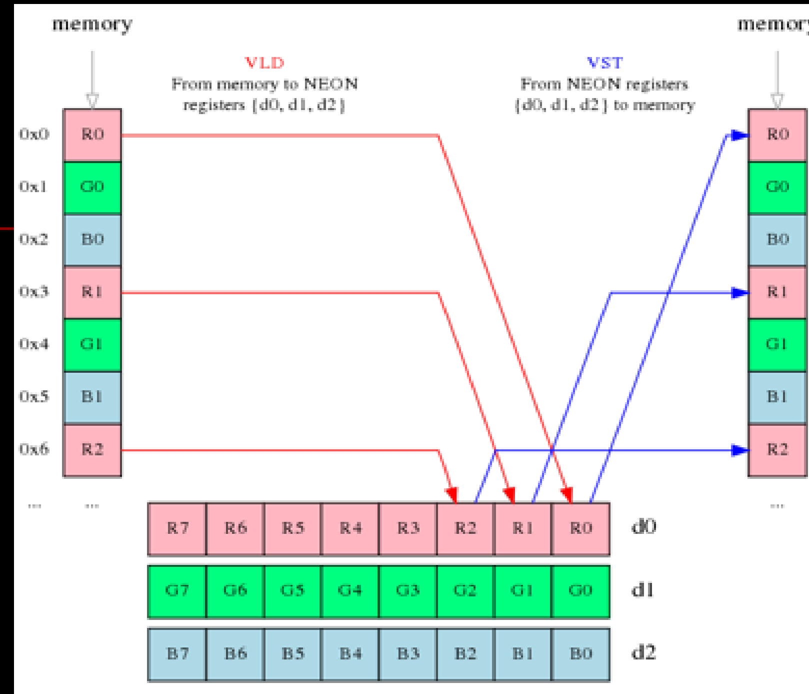
```
#include <stdio.h>
#include <arm_neon.h>

unsigned short int A[] = {1,2,3,4}; // array with 4 elements

int main(void)
{
    uint16x4_t v; // declare a vector of four 16-bit lanes

    v = vld1_u16(A); // load the array from memory into a vector
    v = vadd_u16(v,v); // double each element in the vector
    vst1_u16(A, v); // store the vector back to memory

    return 0;
 }
```

- Unsigned integer U8 U16 U32 U64.
- Signed integer S8 S16 S32 S64.
- Integer of unspecified type I8 I16 I32 I64.
- Floating-point number F16 F32.
- Polynomial over {0,1} P8.

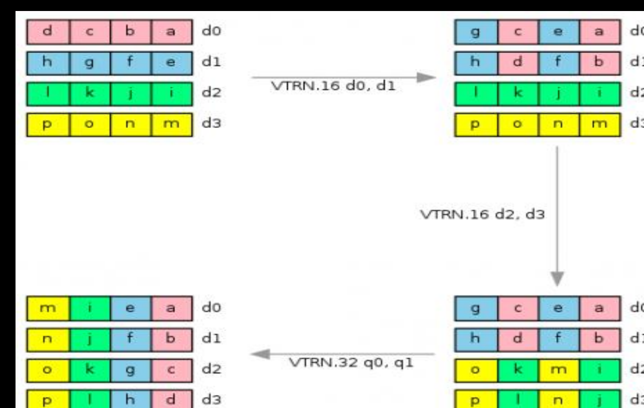
Unroll the loop to the appropriate number of iterations and perform other transformations such as using pointer.

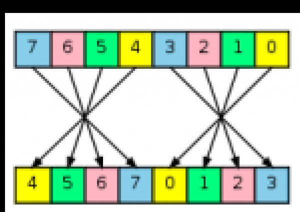
```
void add_int (int * __restrict pa,  int* __restrict pb,  unsigned n,  int x )
{
  unsigned int i ;
  for   ( i  = (( n & ~3 )>>2 ) ; i ; i -- )
  {
     *( pa + 0  ) = *(  pb + 0  ) + x ;
     *( pa + 1  ) = *(  pb + 1  ) + x ;
     *( pa + 2  ) = *(  pb + 2  ) + x ;
     *( pa + 3  ) = *(  pb + 3  ) + x ;
      pa  += 4 ;  pb  += 4 ;
  }
              }
```



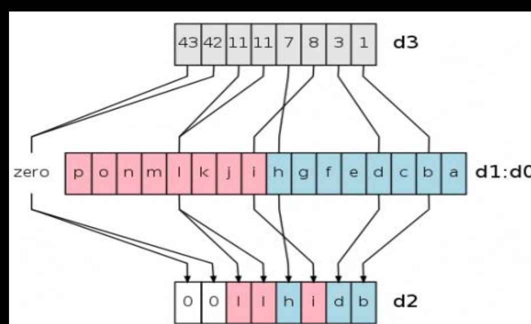


NEON structure loads and stores.





VREV32.8 d0, d1



VTBL d2, {d0, d1}, d3

Transposing a 4x4 matrix

III. HW Implementation

1) Instruction Set Extension

- Intel Polynomial Multiplication Instruction
- http://en.wikipedia.org/wiki/CLMUL_instruction_set

PCLMULQDQ Instruction Definition

PCLMULQDQ instruction performs carry-less multiplication of two 64-bit quadwords which are selected from the first and the second operands according to the immediate byte value.

Instruction format: PCLMULQDQ xmm1, xmm2/m128, imm8

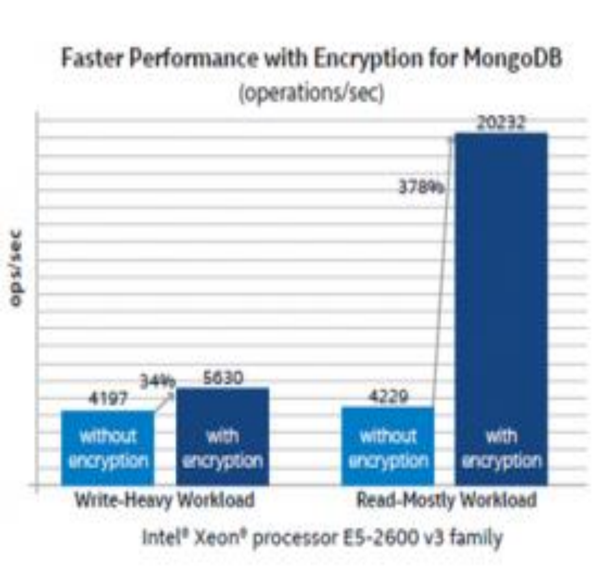
Description: Carry-less multiplication of one quadword (8 bytes) of xmm1 by one quadword (8 bytes) of xmm2/m128, returning a double quadword (16 bytes). The immediate byte is used for determining which quadwords of xmm1 and xmm2/m128 should be used.

Opcode: 66 0f 3a 44

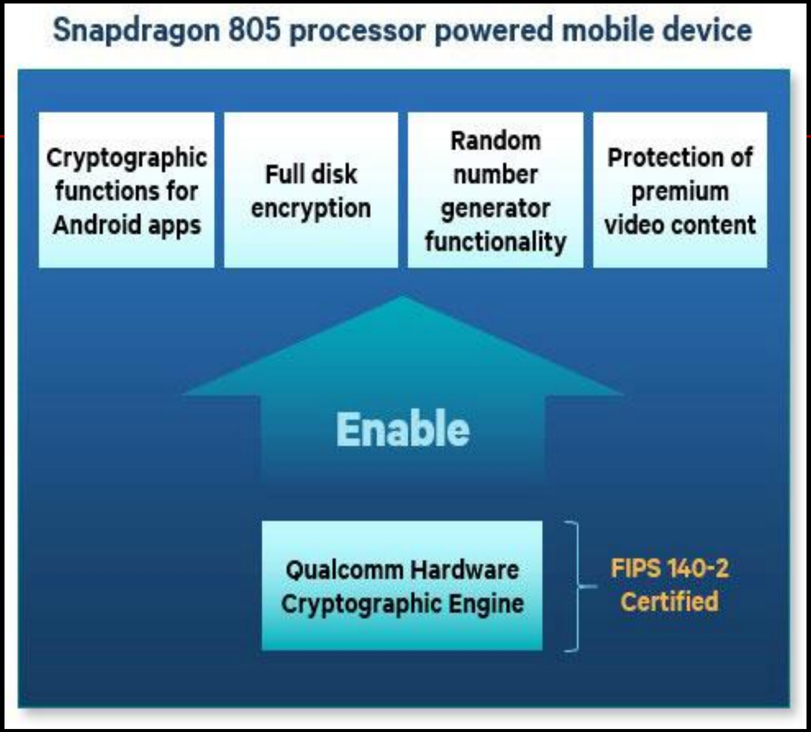
The presence of PCLMULQDQ is indicated by the CPUID leaf 1 ECX[1].

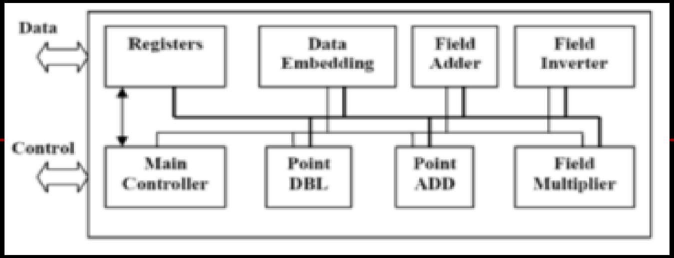
Operating systems that support the handling of Intel SSE state will also support applications that use AES extensions and the PCLMULQDQ instruction. This is the same requirement for Intel SSE2, Intel SSE3, Intel SSSE3, and Intel SSE4.

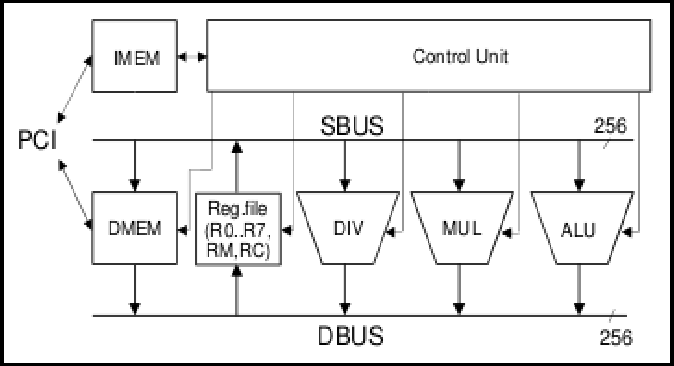
AES-NI



2) Crypto Coprocessor/Accelerator



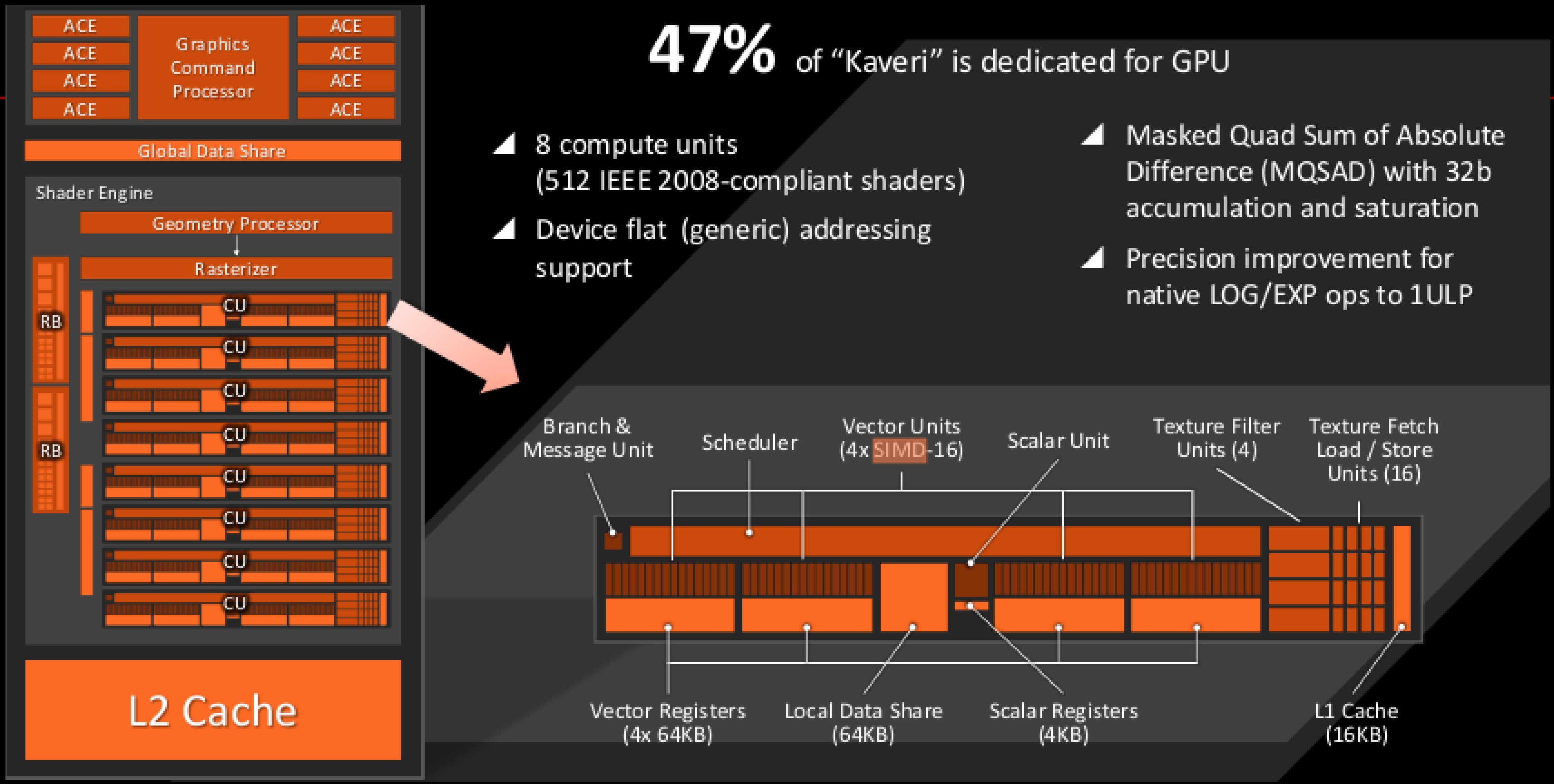




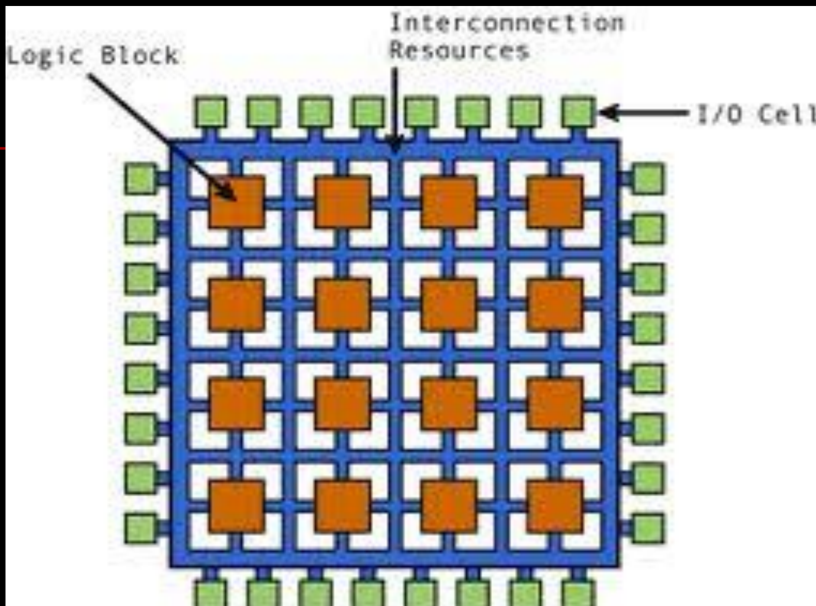
- ASIC
- ASIP

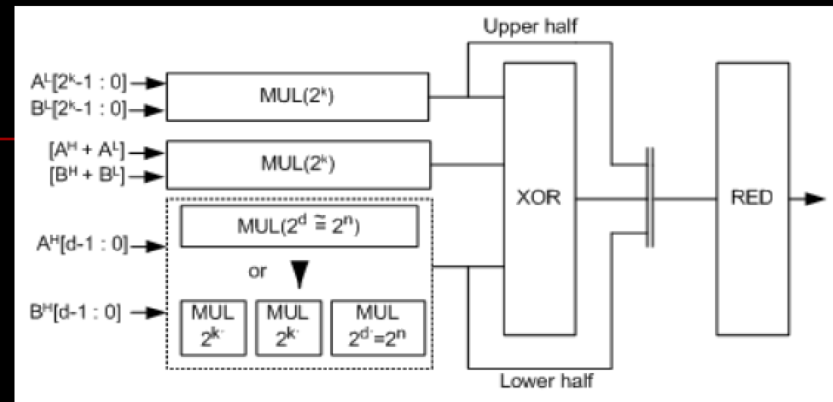
Scalability?

3) GPGPU

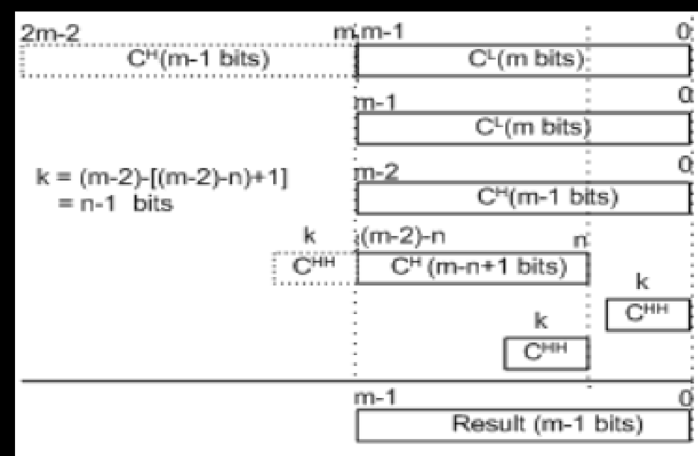

ACE
ACE
ACE
ACE
Graphics Command Processor
ACE
ACE
ACE
ACE
Global Data Share
Shader Engine
Geometry Processor
Rasterizer
RB
RB
CU
CU
CU
CU
CU
CU
CU
CU
L2 Cache
47% of "Kaveri" is dedicated for GPU
8 compute units (512 IEEE 2008-compliant shaders)
Device flat (generic) addressing support
Masked Quad Sum of Absolute Difference (MQSAD) with 32b accumulation and saturation
Precision improvement for native LOG/EXP ops to 1ULP
Branch & Message Unit
Scheduler
Vector Units (4x SIMD-16)
Scalar Unit
Texture Filter Units (4)
Texture Fetch Load / Store Units (16)
Vector Registers (4x 64KB)
Local Data Share (64KB)
Scalar Registers (4KB)
L1 Cache (16KB)

4) FPGA





Block diagram for binary Karatsuba Multiplier



Reduction Diagram

IV. Application

1) Bitcoin

- https://bitcoin.org/en/developer-guide

Bob must first generate a private/public key pair before Alice can create the first transaction. Bitcoin uses the Elliptic Curve Digital Signature Algorithm (ECDSA) with the secp256k1 curve; secp256k1 private keys are 256 bits of random data. A copy of that data is deterministically transformed into an secp256k1 public key. Because the transformation can be reliably repeated later, the public key does not need to be

The elliptic curve domain parameters over $\mathbb{F}_p$ associated with a Koblitz curve secp256k1 are specified by the sextuple $T = (p, a, b, G, n, h)$ where the finite field $\mathbb{F}_p$ is defined by:

$$\begin{aligned} p &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F} \\ &= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 \end{aligned}$$

The curve E: $y^2 = x^3 + ax + b$ over $\mathbb{F}_p$ is defined by:

$$\begin{aligned} a &= \texttt{00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000} \\ b &= \texttt{00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000007} \end{aligned}$$

The base point G in compressed form is:

$$G = \texttt{02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798}$$

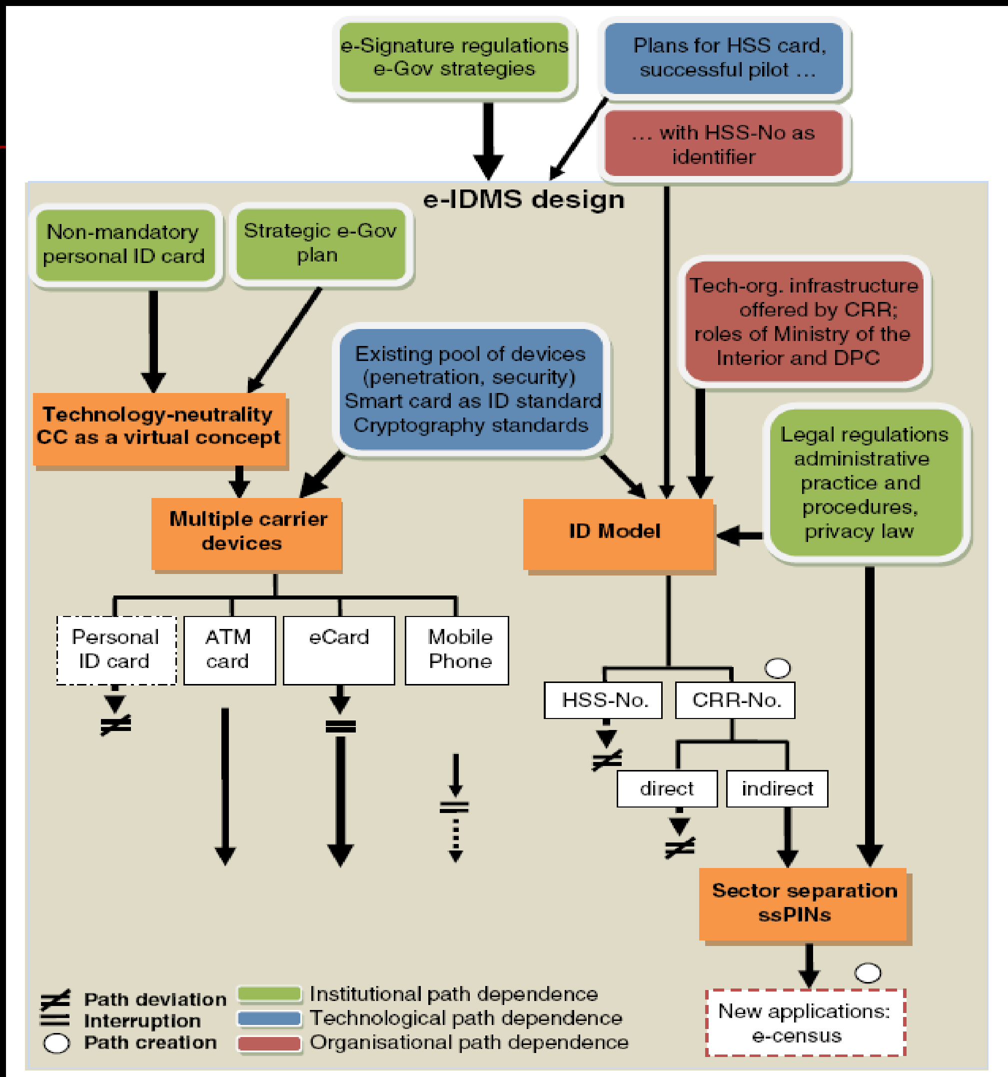
and in uncompressed form is:

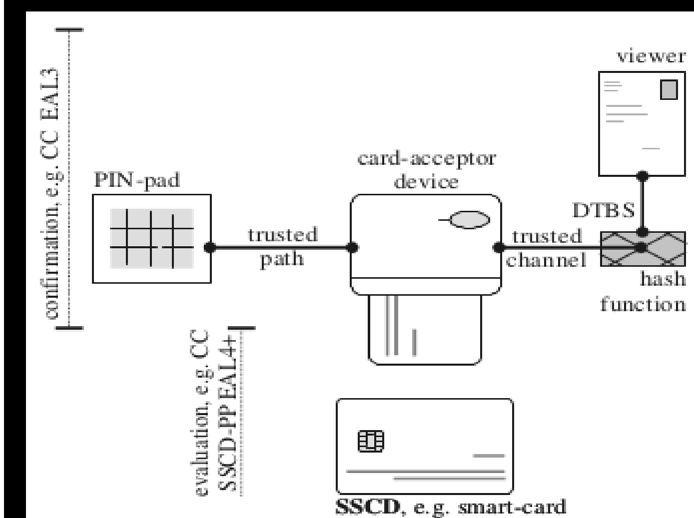
$$G = \texttt{04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8}$$

Finally the order n of G and the cofactor are:

$$\begin{aligned} n &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141} \\ h &= \texttt{01} \end{aligned}$$

2) e-IDMS (Europe)





V. Reference

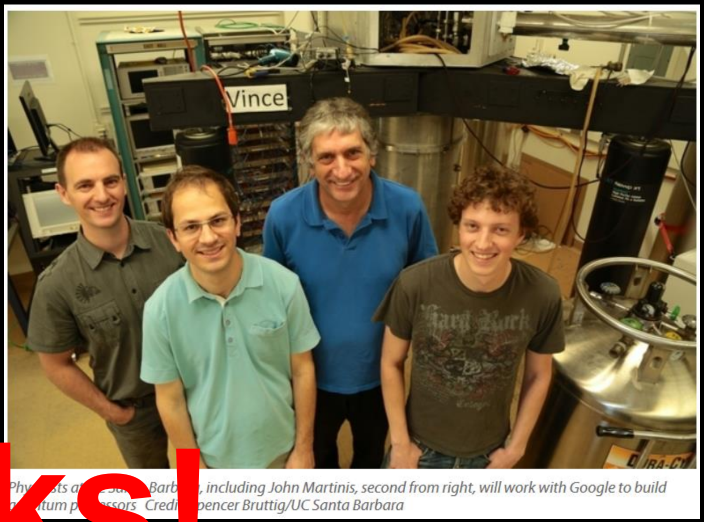
- http://en.wikipedia.org/wiki/Wiki
- http://en.wikipedia.org/wiki/Elliptic_curve_cryptography

Q & A

单击添加文字

Thanks!


The D-Wave Two™ System
The first commercial quantum computer


Vince
including John Martinis, second from right, will work with Google to build
pencer Bruttig/UC Santa Barbara


September 2013
ars technica
MAIN MENU
MY STORIES: 25
FORUMS
SUBSCRIBE
JOBS
RISK ASSESSMENT / SECURITY & HACKTIVISM
Stop using NSA-influenced products, RSA tells custome
Firm "strongly recommends" customers stop using RNG backdoor.
Deliberately flawed? RSA Security tells customers to drop NSA-related encryption algorithm
theguardian
News | US | World | Sports | Comment | Culture | Business | Money
Data
News › World news › NSA
Major US security company warns over NSA link to encryption formula
RSA, the security arm of EMC, sends email to customers over default random number generator which uses weak formula

添加文