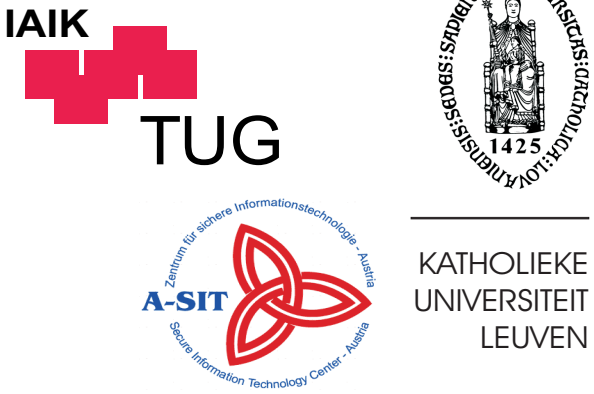
Secure and Efficient Implementations of Elliptic Curve Cryptosystems



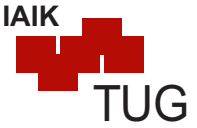
Elisabeth Oswald

Outline

IAIK TUG

- **First Part**
  - Motivation
  - Implementation Options Overview
- **Second Part**
  - Finite Field Arithmetic
  - Elliptic Curve Arithmetic
- **Third Part**
  - Security Issues

Why use ECC?



There are two main reasons:

Security: We need a fallback system which we can use in case factoring becomes too easy.

Efficiency: RSA key sizes are increasing rapidly. Efficient implementations on devices with limited resources are becoming more difficult.

$\Rightarrow$ EC Cryptosystems seem to exactly fit the requirements of state-of-the-art cryptographic applications at the moment:

- key size: is much smaller than for IFP-based cryptosystems since the best known algorithm for solving the ECDLP is exponential in the number of key bits.
- signature length: due to short keys and pre-defined sets of good EC, the signature sizes are considerably shorter compared to systems based on IFP or DLP.
- implementation constraints: due to the short keys special purpose hardware is not necessarily needed.

The probably most important ECC: the ECDSA – 1

IAIK TUG

Suppose Alice wants to send a digitally signed message to Bob. They first choose a finite field $\mathbb{F}_q$, an elliptic curve E, defined over that field and a base point G with order n. Alice's key pair is (d, Q), where d is her private and Q is her public key. To sign a message M Alice does the following:

1.	Choose a random number k with $k : 1 \leq k \leq n-1$.
2.	Compute $kG = (x_1, y_1)$ and $r = x_1 \bmod n$. If $r = 0$ then go to step 1.
3.	Compute $k^{-1} \bmod n$.
4.	Compute $e = \text{SHA-1}(M)$.
5.	Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then go to step 1.
6.	Alice signature for the message M is (r, s).

Figure 1: ECDSA Signature Generation

The probably most important ECC: the ECDSA – 2

IAIK TUG

To verify Alice's signature (r, s) on the message m, Bob obtains an authentic copy of Alice's parameters and public key. Bob should validate the obtained parameters! Bob then does the following:

1.	Verify that r, s are integers in the interval $[1, n-1]$.
2.	Compute $e = \text{SHA-1}(M)$.
3.	Compute $w = s^{-1} \bmod n$.
4.	Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5.	Compute $X = u_1 G + u_2 Q$. If $X = \mathcal{O}$ then reject the signature. Otherwise compute $v = x_1 \bmod n$ where $X = (x_1, y_1)$.
6.	Accept the signature if and only if $v = r$.

Figure 2: ECDSA Signature Verification

Implementation Issues

IAIK TUG

Apparently, we have to implement

- Arithmetic in $\mathbb{F}_n$: reduction, addition/subtraction, multiplication/division
- Arithmetic on the elliptic curve E
- a hash function.

In this talk we do not consider implementations of hash functions, but focus on implementing the arithmetic.

Thus, we can make choices regarding the curve itself, the finite field and the representation of its elements and the representation of the curve points, etc. What we actually choose depends on our needs (application).

Implementation Options – 1

Choice of the underlying field: depending on hard- or software implementations

Representation of the elements of this field: influences mostly the speed

Implementing the arithmetic in the field: the same

Selecting an appropriate curve: influences speed and security

Implementing the EC operations: influences speed and security

The first three points are concerned with the finite field arithmetic while the last two points are concerned with the elliptic curve arithmetic.

Often a choice can lead to weak cryptosystems. Fortunately, there exists a set of (named) curves (published by the NIST) which have been checked.

Implementation Options – 2

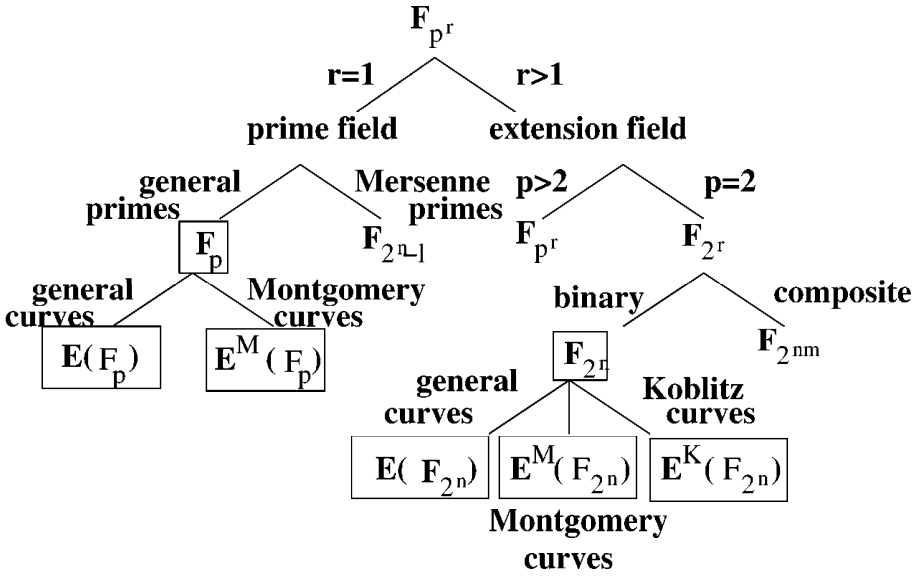


Figure 3: Implementation options – an overview

There are more possibilities to parameterise curves; often these other parameterisations have advantages with respect to side-channel attacks.

(Intermediate) Conclusion

- There exist a plenty of choices regarding implementations
- The most important characteristics of an implementation are typically:
  - Speed
  - Versatility
  - Size (circuit, code)
  - Power Consumption (contactless smart cards)
  - Interoperability, Standards, Patents
  - Security (side-channel attacks, fault attacks)
- It might be dangerous to choose your own curve, thus if you want to play safe, use the NIST curves.

Finite Field Arithmetic

IAIK TUG

- Finite Fields (Galois Fields) with $q = p^m$ elements: $\mathbb{F}_q$ (aka GF(q)).
- Mostly interesting nowadays are only these two:
- GF(p) (prime finite fields), operations are all done modulo a prime number p.
- $GF(2^m)$ (binary extension fields), operations are all done modulo an irreducible polynomial $p(t)$.

The choice of the modulus (and irreducible polynomial, resp.) influences both the security and the efficiency of the whole system. In order to have high security, long moduli have to be used.

The problem is then: Implement arithmetic operations with long numbers (on processors with short word length).

Hardware vs. Software Implementations

IAIK TUG

- **Dedicated Hardware:**
  - Typically one uses a co-processor for the arithmetic operations.
  - It has an n-bit datapath and n-bit registers.
  - Bit-serial or digit-serial multiplier architectures are used.
- **Software Implementations:**
  - Efficient algorithms for multi-precision arithmetic are used.
  - Implementations use word level instructions of the processor.
  - Montgomery Multiplication is popular

RSA implementations on smart cards make typically use of a co-processor while ECC implementations do not.

Any implementation that is not targeted towards a specific type of curve will be potentially big and slow. Any implementation that is specifically adapted for one implementation option is not versatile. Hence, it depends on your application what you can do!

GF(p) – Basic Operations

Remember: $n = \lceil \log_2(p) \rceil$

Reduction: Naive reduction requires either division or subtraction.

Addition/Subtraction: Result can be at most 2(p-1). Subtracting the modulus once is sufficient to get a reduced result.

Division/Inversion: Either use the extended Euclidian Algorithm (painful in hardware, also slow in software) or exponentiation, probably with interleaved reductions.

Multiplication: Montgomery architecture is one of the well studied ones but also Sedlak's algorithm is used (in Infineon's SLE66X ICs).

The general problem with standard modular reduction is that you never know how often you have to subtract the modulus! Also, comparisons are rather inefficient in hardware.

GF(p) – Multiplication in Hardware

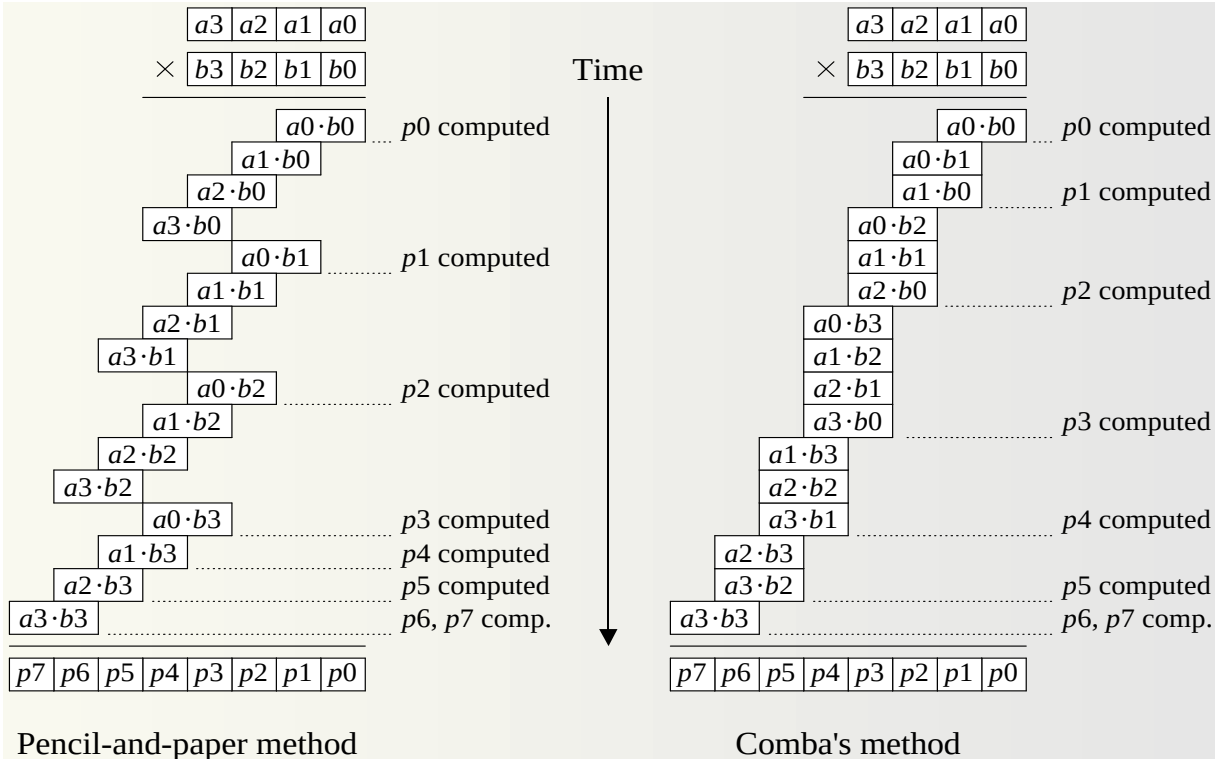
IAIK TUG

Input:Integers $N = (n_{l-1} \cdots n_1 n_0)_2$, $x = (x_l \cdots x_1 x_0)_2$, $y = (y_l \cdots y_1 y_0)_2$ with $x < 2N, y < 2N$, $R = 2^{l+2}$, $gcd(N, 2) = 1$ and $N' = -N^{-1} \mod 2$ **Output:**$xyR^{-1} \mod 2N$
$T \leftarrow 0$
For i from 0 to $l + 1$
$m_i \leftarrow (t_0 + x_i y_0)\, N' \bmod 2$ (with $T = (t_l t_{l-1} \ldots t_0)$)
$T \leftarrow (T + x_i y + m_i N)/2$

Table 1: Montgomery modular multiplication without final subtraction

You need approximately l steps (clock cycles) to calculate the result (for the Radix 2 version). Using systolic arrays or clever (redundant) number representations can avoid too long carry chains.

GF(p) – Multiplication in Software



The result has to be reduced as well $\Rightarrow$ Montgomery reduction is also well suited for software implementations.

GF(p) – Mersenne Numbers

IAIK TUG

- A Mersenne number is of the form: $m = 2^k - 1$
- Integers $\mod m$ are represented k-bit integers
- If $n < m^2$ is the integer that is to be reduced $\mod m$
- Let $n = 2^k T + U$ with T and U being k-bit integers
- Then $n \equiv T + U \mod m$
- Thus, the integer division can be replaced by a modular addition!
- This is very useful when k is a multiple of the word size of a processor.
- Since the wordsize is typically a power of 2, one must choose k which are highly composite
- Mersenne numbers arising from this construction are almost never primes
- Salinas generalized this approach

GF(p) – Generalized Mersenne Numbers

IAIK TUG

- $p = 2^{n_l w} \pm 2^{n_{l-1} w} \pm \cdots \pm 2^{n_1 w} \pm 1$
- $\longrightarrow$ coefficients can be reduced recursively:
- $2^{n_l w} \equiv \pm 2^{n_{l-1} w} \pm \cdots \pm 2^{n_1 w} \pm 1.$
- The paper of Solinas gives a couple of conditions for such numbers:
- An integral polynomial is suitable for Mersenne reduction if it is
  - reduced, proper, irreducible, of low weight and has an odd constant term.
- A couple of examples are given.
- The following generalized Mersenne primes appear in the document **Recommended Elliptic Curves for Federal Government use** which is provided by the NIST:
  - $p = 2^{192} - 2^{64} - 1,$
  - $p = 2^{224} - 2^{96} + 1,$
  - $p = 2^{256} - 2^{244} + 2^{196} + 2^{96} - 1,$
  - $p = 2^{384} - 2^{128} - 2^{96} - 1.$

GF(p) – Example

- We look at generalised Mersenne primes of the form $p = 2^{3k} - 2^k - 1$
- The polynomial $f(t) = t^3 - t - 1$ generates such numbers (set $t = 2^k$).
- The integers $\mod p$ can be represented as $3k$-bit integers and each integer $n < p^2$ as a $6k$-bit expression $n = \sum_{j=0}^{5} A_j 2^{jk}$, with $0 \leq A_j < 2^k$.
- We'd like to find $B_j's$ such that $n \equiv \sum_{j=0}^{2} B_j 2^{jk} \pmod p$.
- We know that
  - $t^3 \equiv t + 1 \pmod{f(t)}$
  - $t^4 \equiv t^2 + t \pmod{f(t)}$
  - $t^5 \equiv t^2 + t + 1 \pmod{f(t)}$
- It follows that $\sum_{j=0}^{5} A_j t^j = (A_0 A_1 A_2) \begin{pmatrix} 1 \\ t \\ t^2 \end{pmatrix} + (A_3 A_4 A_5) \begin{pmatrix} t^3 \\ t^4 \\ t^5 \end{pmatrix}$, and
- $(B_0 B_1 B_2) = \left((A_0 A_1 A_2) + (A_3 A_4 A_5) \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \right) \begin{pmatrix} 1 \\ t \\ t^2 \end{pmatrix} \mod f(t)$

GF(p) – Example (cont.)

- Rewriting the equation from the last slide we obtain:

$$\begin{aligned} B_0 &= A_0 + A_3 + A_5 \\ B_1 &= A_1 + A_3 + A_4 + A_5 \\ B_2 &= A_2 + A_4 + A_5 \end{aligned}$$

- By rearranging this nicely we get: $B = T + S_1 + S_2 + S_3 \mod p$ with

$$\begin{aligned} T &= (\; A_2 \; \| \; A_1 \; \| \; A_0 \;) \\ S_1 &= (\; 0 \; \| \; A_3 \; \| \; A_3 \;) \\ S_2 &= (\; A_4 \; \| \; A_4 \; \| \; 0 \;) \\ S_3 &= (\; A_5 \; \| \; A_5 \; \| \; A_5 \;) \end{aligned}$$

$\Rightarrow$ modular reduction corresponds to 3 modular additions.

GF(2^m) – Basics

IAIK TUG

- Elements are represented as binary polynomials: $A = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$ with $a_i \in \{0, 1\}$.
- Calculations are done modulo an irreducible polynomial $p(x)$.
- Arithmetic in this field is well studied since 1960s due to applications in coding.
- Arithmetic is heavily influenced by choice of basis
- Bases which have been proposed for applications:
  1. polynomial basis (standard basis)
  2. normal basis
  3. dual basis, triangular basis, etc.. . .
- Most applications represent elements by a polynomial basis

GF(2^m) – Basis Representations

IAIK TUG

- **Polynomial Basis:** Best use a *trinomial* basis. This is a representation in which the irreducible polynomial has the form $x^m + x^k + 1$. Such representations have the advantage that reduction modulo $p(x)$ can be performed efficiently, both in software and in hardware.
- **Normal Basis:** A normal basis of $GF(2^m)$ over $GF(2)$ is a basis of the form $\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\}$. Squaring is a linear operation in $GF(2^m)$, so, $A^2 = \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}} = \sum_{i=0}^{m-1} a_{i-1} \beta^{2^i} = (a_{m-1}, a_0, \ldots, a_{m-2})$. Thus, squaring is just a simple rotation in this basis representation. This is especially nice in hardware. Multiplication is more complicated.
- **Optimal Normal Basis:** They are special types of normal basis representations which allow also efficient multiplication.

Another advantage of normal basis is that square roots of elements in $GF(2^m)$ can be computed efficient. For some point compression techniques, recovering the points need that operation.

GF(2^m) – Basic Operations

IAIK TUG

Addition: simple XOR operation, no carry

Multiplication: use (variants) of shift-and-xor

Squaring: is a linear operation! $A = (\sum_{m=0}^{m-1} a_i x^i)^2 = \sum_{m=0}^{m-1} (a_i)^2 x^i$

Inversion: use EEA or exponentiation ($\Rightarrow$ try to avoid this operation)

Reduction: is very fast when special irreducibles are used (trinomials or pentanomials)

GF(2^m) – Bit-serial architecture

IAIK TUG

Input:binary polynomials $A(x)$, $B(x)$ of degree $n-1$
Output:$C(x) = A(x) * B(x)$
FOR $i = n-1$ DOWNTO 0
C(x) = x*C(x) . . . 1-bit shift
IF $b_i == 1$ THEN C(x)=C(x) XOR A(x)

Table 2: Shift and XOR

Nice to implement in hardware but not so nice for software! Also, various improvements are possible $\Rightarrow$ windowing methods!

Performance Comparison

IAIK TUG

"Performance comparisons of elliptic curve systems in software" by *K. Fong, D. Hankerson, J. López, A. Menezes*, and *M. Tucker*. Presented at the 5th ECC Workshop, Oct. 2001.

Operation	Prime fields GF(p)			Binary fields GF(2^m)		
	|p|=192	|p|=224	|p|=256	m=163	m=233	m=283
Addition	0.055	0.062	0.071	0.032	0.039	0.041
Fast Reduction	0.097	0.122	0.256	0.081	0.094	0.145
Multiplication	0.350	0.456	0.681	1.058	1.923	2.403
Squaring	0.300	0.394	0.600	0.185	0.238	0.312
Inversion	21.1	28.4	36.8	10.0	17.4	24.5

- Timings in µs (1000 MHz Pentium III), C and Assembler.
- Multiplication in GF(2^m): Shift-and-XOR, window-size: 4.

(Intermediate) Conclusion

IAIK TUG

- **Prime fields for software implementations:**
  - Fast when machine has a hardware multiplier
  - Very flexible (various optimisations are possible)
- **Binary Extension fields for hardware implementations:**
  - Simple to design (no carry propagation)
  - Require less HW than multiplier
  - Low power consumption (important for smart cards)

Elliptic Curve Arithmetic

IAIK TUG

An *elliptic curve* E over the field $\mathbb{F}$ is a smooth curve in the so called "long Weierstrassform"

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \quad a_i \in \mathbb{F}. \tag{1}$$

We let $E(\mathbb{F})$ denote the set of points $(x, y) \in \mathbb{F}^2$ that satisfy this equation, along with a "point at infinity" denoted $\mathcal{O}$.

An elliptic curve E over the finite field $\mathbb{F}_p$ is given through an equation of the form

$$y^2 = x^3 + ax + b, \quad a, b \in \mathbb{F}_p, \text{ and } -(4a^3 + 27b^2) \neq 0 \tag{2}$$

A (nonsupersingular) elliptic curve E over the finite field $\mathbb{F}_{2^m}$ is given through an equation of the form

$$y^2 + xy = x^3 + ax^2 + b, \quad a, b \in \mathbb{F}_{2^m}. \tag{3}$$

IAIK TUG

Addition Law – Affine Coordinates

	$\mathbb{F}_p$	$\mathbb{F}_{2^m}$
P+Q	$x_3 = (\frac{y_2-y_1}{x_2-x_1})^2 - x_1 - x_2$	$x_3 = (\frac{y_1+y_2}{x_1+x_2})^2 + \frac{y_1+y_2}{x_1+x_2} + x_1 + x_2 + a$
	$y_3 = (\frac{y_2-y_1}{x_2-x_1})(x_1 - x_3) - y_1$	$y_3 = (\frac{y_1+y_2}{x_1+x_2})(x_1 + x_3) + x_3 + y_1$
2P	$x_3 = ((3x_1^2 + a)/2y_1)^2 - 2x_1$	$x_3 = x_1^2 + \frac{b}{x_1^2}$
	$y_3 = -y_1 + ((3x_1^2 + a)/2y_1)(x_1 - x_3)$	$y_3 = x_1^2 + (x_1 + \frac{y_1}{x_1})x_3 + x_3$
-P	$(x, -y)$	$(x, x + y)$

Table 3: Addition formulas in affine coordinates

Note the many inversions that need to be calculated!

Projective Space

Definition 1. *The projective space is the set of equivalence classes of tuples* $(X_0, X_1, \ldots, X_n)$ *(not all components zero) where two tuples are said to be equivalent if they are scalar multiples of one another, i.e.*

$$\mathbb{P}^n(k) = \{(X_0, X_1, \ldots, X_n) - (0, 0, \ldots, 0) | (tX_0, tX_1, \ldots, tX_n) \sim (X_0, X_1, \ldots, X_n), t \in \mathbb{Z}\backslash\{0\}.\}$$

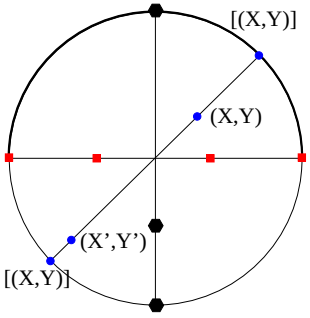


Figure 4: Projective Line

Addition Law – Projective Coordinates

The equation for an EC over $F_q, char > 3$ in projective coordinates, corresponding to the change $x = X/Z$ and $y = Y/Z$ is $Y^2Z = X^3 + aXZ^2 + bZ^3$. If $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, Z_2)$ are points on the EC then:

$$
\begin{aligned}
X_3 &= P_1(-Q_2 + T) \\
2Y_3 &= R(-2T + 3Q_2) - P_3(S_2 + S_1) \\
Z_3 &= WP_3
\end{aligned}
$$

$$
\begin{aligned}
U_1 &= X_1Z_2 \\
U_2 &= X_2Z_1 \\
S_1 &= Y_1Z_2 \\
S_2 &= Y_2Z_1 \\
W &= Z_1Z_2 \\
P_1 &= U_2 - U_1
\end{aligned}
\quad \text{and} \quad
\begin{aligned}
P_2 &= P_1^2 \\
P_3 &= P_1P_2 \\
Q_1 &= U_1 + U_2 \\
Q_2 &= P_2Q_1 \\
R &= S_2 - S_1 \\
T &= WR^2
\end{aligned}
$$

Figure 5: Point Addition

$$
\begin{aligned}
X_3 &= \lambda_1(\lambda_4 - 4\lambda_5) \\
Y_3 &= -2Y_1^2\lambda_6 + \lambda_2(6\lambda_5 - \lambda_4) \\
Z_3 &= \lambda_1\lambda_6
\end{aligned}
$$

$$
\begin{aligned}
\lambda_1 &= 2Z_1Y_1 \\
\lambda_2 &= 3X_1^2 + aZ_1^2 \\
\lambda_3 &= X_1Y_1
\end{aligned}
\quad \text{and} \quad
\begin{aligned}
\lambda_4 &= \lambda_2^2 \\
\lambda_5 &= \lambda_1\lambda_3 \\
\lambda_6 &= \lambda_1^2
\end{aligned}
$$

Figure 6: Point Doubling

Addition Law – Projective Coordinates

Operation (Basis Field F_p)	Coordinates		
	affine	projective $x = X/Z$ $y = Y/Z$	weighted projective $x = X/Z^2$ $y = Y/Z^3$
Addition	1I+3M	14M	16M
Doubling	1I+4M	13M	10M

Weighted projective coordinates (or Jacobian coordinates) are used in the Annex A of the IEEE 1363 standard. However, several other coordinate systems do exist.

Jacobi Parametrisation – 1

IAIK TUG

For elliptic curves over a field with characteristic > 3, the so called Jacobi Form can be defined: The curve can be seen as intersection of two quadrics ($S^2 + D^2 = T^2$, $k^2S^2 + T^2$). A point is then a four-tuple (S, C, D, T).
Let $E : y^2 = x^3 + ax + b$ an elliptic curve that has three points of order two. By applying a transform that moves the three points of order two to $(0, 0), (-1, 0), (-\lambda, 0)$, we obtain an isomorphic curve $E' : y^2 = x(x + 1)(x + \lambda)$.
Let $\lambda = 1 - k^2$ and let (X, Y, Z) be a point on $E'' : Y^2Z = X(X + Z)(X + \lambda Z)$. Then a point is given by:

$$\begin{aligned} S &= -2(X + Z)Y \\ T &= \lambda(X^2 + Z^2 + 2XZ) + Y^2 \\ C &= \lambda(-X^2 - Z^2 - 2XZ) + Y^2 \\ D &= \lambda Z^2 + Y^2 + 2XZ + (2 - \lambda)X^2 \end{aligned}$$

Jacobi Parametrisation – 2

Take two points $P_1 = (S_1, C_1, D_1, T_1)$ and $P_2 = (S_2, C_2, D_2, T_2)$, the sum $P_3 = (S_3, C_3, D_3, T_3)$ is defined as:

$$\begin{array}{l} S_3 = A_1B_1 + A_2B_2 \\ C_3 = A_1A_2 - B_1B_2 \end{array} \quad \text{and} \quad \begin{array}{l} D_3 = T_1D_1T_2D_2 - k^2S_1C_1S_2C_2 \\ T_3 = (A_1)^2 + (B_2)^2 \end{array}$$

where

$$\begin{array}{l} A_1 = T_1C_2 \\ A_2 = T_2C_1 \end{array} \quad \text{and} \quad \begin{array}{l} B_1 = S_1D_2 \\ B_2 = S_2D_1 \end{array}$$

Figure 7: Point Doubling

Apparently, there is no difference between double and add!

Hesse Parametrisation

We only have a look at this parametrisation in projective coordinates. An important feature of this parametrisation is that it only depends on one parameter: $X^3+Y^3+Z^3 = dXYZ$. For two non-equal points $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, Z_2)$ on the curve, the sum $P_3 = (X_3, Y_3, Z_3)$ is given by:

Input: $\boldsymbol{P} = (U_1 : V_1 : W_1)$ **and** $\boldsymbol{Q} = (U_2 : V_2 : W_2)$ **with** $\boldsymbol{P} \neq \boldsymbol{Q}$
Output: $\boldsymbol{P} + \boldsymbol{Q} = (U_3 : V_3 : W_3)$

$T_1 \leftarrow U_1$; $T_2 \leftarrow V_1$; $T_3 \leftarrow W_1$ $T_4 \leftarrow U_2$; $T_5 \leftarrow V_2$; $T_6 \leftarrow W_2$
$T_7 \leftarrow T_1 \cdot T_6$ $(= U_1W_2)$
$T_1 \leftarrow T_1 \cdot T_5$ $(= U_1V_2)$
$T_5 \leftarrow T_3 \cdot T_5$ $(= W_1V_2)$
$T_3 \leftarrow T_3 \cdot T_4$ $(= W_1U_2)$
$T_4 \leftarrow T_2 \cdot T_4$ $(= V_1U_2)$
$T_2 \leftarrow T_2 \cdot T_6$ $(= V_1W_2)$
$T_6 \leftarrow T_2 \cdot T_7$ $(= U_1V_1W_2^2)$
$T_2 \leftarrow T_2 \cdot T_4$ $(= V_1^2U_2W_2)$
$T_4 \leftarrow T_3 \cdot T_4$ $(= V_1W_1U_2^2)$
$T_3 \leftarrow T_3 \cdot T_5$ $(= W_1^2U_2V_2)$
$T_5 \leftarrow T_1 \cdot T_5$ $(= U_1W_1V_2^2)$
$T_1 \leftarrow T_1 \cdot T_7$ $(= U_1^2V_2W_2)$
$T_1 \leftarrow T_1 - T_4$; $T_2 \leftarrow T_2 - T_5$; $T_3 \leftarrow T_3 - T_6$
$U_3 \leftarrow T_2$; $V_3 \leftarrow T_1$; $W_3 \leftarrow T_3$

Figure 8: Point Addition

Proposition 2. *Let $\boldsymbol{P} = (U_1 : V_1 : W_1)$ be a point on an Hessian elliptic curve $E_D(\mathbb{K})$. Then*

$$2(U_1 : V_1 : W_1) = (W_1 : U_1 : V_1) + (V_1 : W_1 : U_1) \ . \tag{13}$$

Furthermore, we have $(W_1 : U_1 : V_1) \neq (V_1 : W_1 : U_1)$.

Figure 9: Point Doubling

Montgomery Parametrisation

The equation of the curve in projective coordinates is given by $E : bY^2Z = X^3 + aX^2Z + XZ^2$. For two points $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, Z_2)$ the sum $P_3 = (X_3, Y_3, Z_3)$ is defined as:

$$
\begin{aligned}
X_3 &= PQ(W_2 - W_1) \\
Y_3 &= Q(RW_1 + U_2T_1 - U_1T_2) \\
Z_3 &= P^2(T_1 + T_2)
\end{aligned}
$$

$$
\begin{aligned}
U_1 &= X_1Z_2 \\
U_2 &= X_2Z_1 \\
P &= U_2 - U_1 \\
S_1 &= Y_1Z_2 \\
S_2 &= Y_2Z_1 \\
R &= S_2 - S_1
\end{aligned}
\quad \text{and} \quad
\begin{aligned}
T_1 &= Y_1X_2 \\
T_2 &= Y_2X_1 \\
Q &= T_2 - T_1 \\
W_1 &= Z_1Z_2 \\
W_2 &= X_2X_1
\end{aligned}
$$

Figure 10: Point Addition

$$
\begin{aligned}
X_3 &= H(2K(X_1 - Q) + \tilde{P}) \\
Y_3 &= 2K(PQ - K) - P\tilde{P} \\
Z_3 &= H^2W
\end{aligned}
$$

$$
\begin{aligned}
N &= aZ_1 \\
P &= 2NX_1 + Z_1^2 + 3X_1^2 \\
\tilde{P} &= P^2 \\
W &= 2Y_1Z_1
\end{aligned}
\quad \text{and} \quad
\begin{aligned}
H &= bW \\
Q &= N + 3X_1 \\
K &= HY_1
\end{aligned}
$$

Figure 11: Point Doubling

For all curves in Montgomery representation we can find an isomorphic elliptic curve in Weierstrassform. The opposite is not true! Especially for the NIST curves, no Montgomery form does exist.

Koblitz Curves

IAIK TUG

- Defined over $GF(2^n)$, $E : y^2 + xy = x^3 + ax^2 + 1$ with $a \in \{0, 1\}$.
- Due to the special choice of the parameters, if (x, y) is a point on the curve, so is (x^2, y^2).
- Even more, one can show that $\tau(x, y) = (x^2, y^2)$ with $\tau^2 - (-1)^{1-\alpha}\tau + 2 = 0$.
- An important feature of this relationship is as follows. If the scalar k (of kP) is represented with radix τ, then in the computation of kP, the operation $Q = 2Q$ is replaced by $Q = \tau Q$.
- The latter corresponds to two squaring operations over $GF(2^n)$.
- Scalar multiplication algorithms can make use of this property.

Scalar Point Multiplication

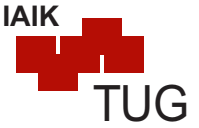
- Oldest and simplest method is the binary algorithm. It can be performed bottom-up or top-down.
- A direct extension is the m-ary method.
- Window methods process windows up to a specific size.
- Due to the fact that subtractions on an elliptic curve are not more expensive than additions, signed digit representations can be used as well.
- The most efficient signed digit representations (i.e. the sparse ones) are called NAF (non-adjacent form).
- Of course, window or m-ary methods can be used together with the signed digit representations.

(Intermediate) Conclusion

IAIK TUG

- Affine Coordinates lead to nice mathematical descriptions for the addition and the doubling operation, but projective coordinates do not require to compute field inversions. Hence, they are better suited for implementations.
- Elliptic curves in Weierstrass form are the "standard" curves. There are shorter formulas for EC which are defined over $GF(p)$ and $GF(2^m)$. Besides the Weierstrass curves also Koblitz curves are included in the set of curves which have been recommended by the NIST.
- There are many other forms (parametrisations) of EC. All have their own advantages and disadvantages (especially with respect to secure implementations).
- There exist many different types of point multiplication algorithms. They differ mainly in speed and in resistance against implementation attacks. Due to the fact that the subtraction of a point is not costly, signed digit representations can be used to achieve further speedup.

Security Issues – Implementation Attacks



- Conventional cryptanalysis treats cryptographic algorithms as purely mathematical objects.
- Side-channel cryptanalysis also takes the implementations of the algorithms into account.
- Peter Wright (MI5) reports several occasions when MI5 performed side-channel analysis to break ciphers (ENGULF, STOCKADE)
- So far, several types of side-channels have been used:
  - Sound
  - Execution Time
  - Power Consumption
  - Electromagnetic Emanations
  - Error Messages
- Combinations of two or more side-channels are possible, but have not been intensively investigated.
- Fault Attacks are active implementation attacks.

Simple Side-Channel Attacks

- Side-channel output is mainly depending on the performed operations.
- Typically, a single *trace* is used in the analysis.
- Mostly, the secret key can be directly read from the side-channel trace.
- A simple example: Unconcealed double and add operations in ECC directly reveal the ephemeral keys.
- Of course, exceptions to these assumptions do exist . . .

Differential Side-Channel Attacks

- Side-channel output is mainly depending on the performed data.
- Typically, many traces are used in the analysis.

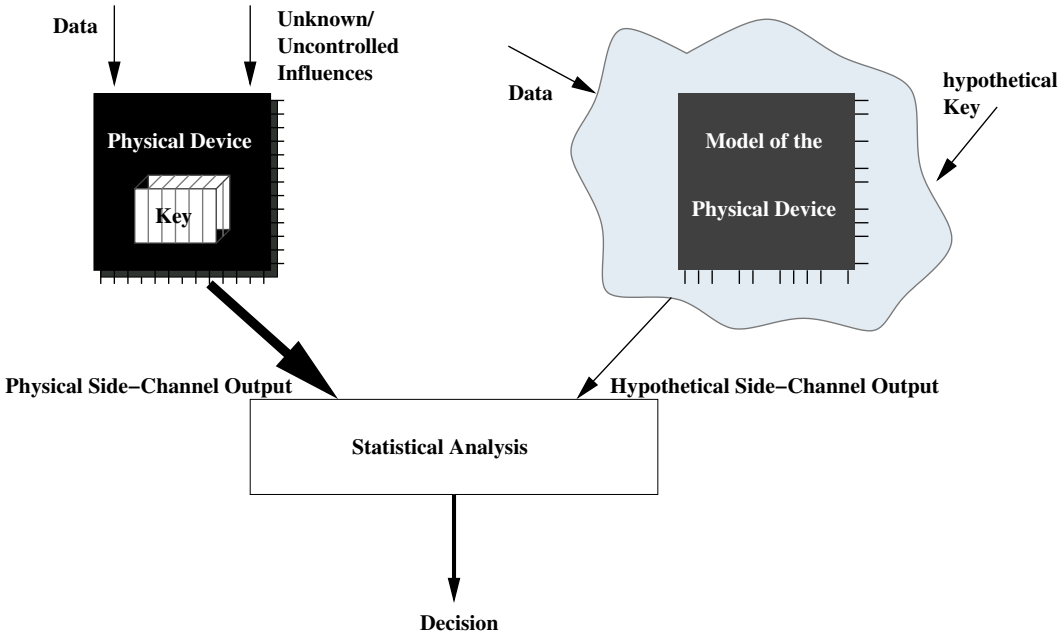
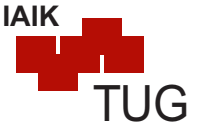


Figure 12: DSCA

Types of Countermeasures



Hardware Countermeasures: such as the use of special types of logic or masking methods. The amount of information leakage can be reduced. Unfortunately these approaches result in expensive implementations.

Software Countermeasures: such as blinding (masking) or randomisation methods. The statistical correlation between the executed data and the side-channel can be reduced.

Protocol Countermeasures: such as frequently changing of the key material. In this way, an attacker can never obtain recent key material.

Generic Countermeasures: such as inserting random delays between consecutive operations. They can be implemented in hardware or software, regardless of the cryptographic algorithm.

Decent implementations **never** rely on a single countermeasure!

Analysing an EC Hardware Implementation

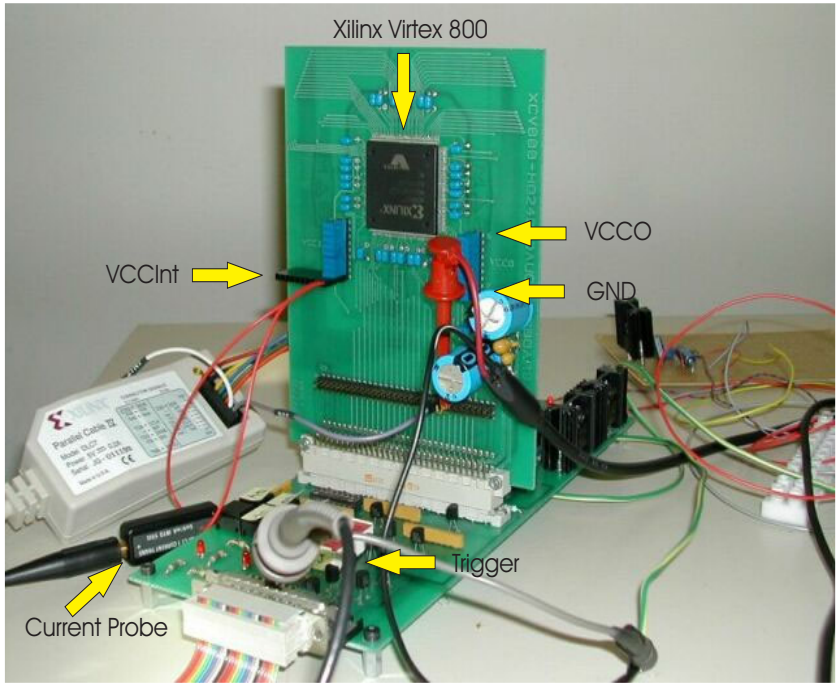


Figure 13: The setup

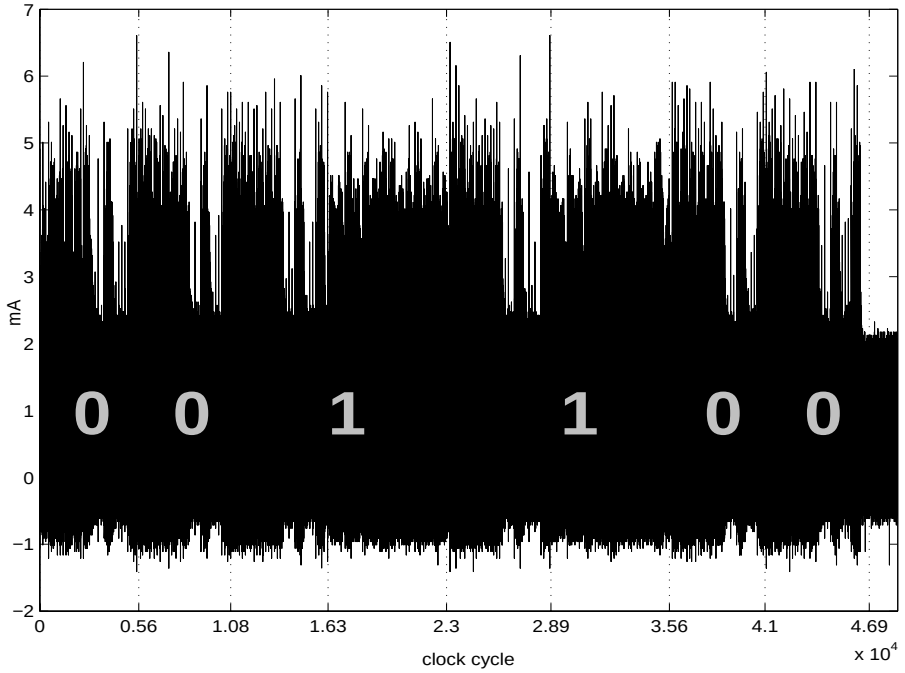


Figure 14: EC Double and Add

SCA on Public-Key Cryptosystems

There are three target operations for SCA in PKCs:

- Multiplications,
- Exponentiations, and
- Scalar Point-Multiplications.

The problems in all of the three operations mentioned above are because of the use of some variant of the binary algorithm.

ECCs offer a particularly wide range of implementation options that can be used to secure their implementations against SCAs. One of them is based on fiddling around with point-multiplication algorithm:

- If one uses a double-add-and-subtract algorithm, and an attacker is unable to distinguish adds and subtracts, then an SSCA is not directly possible.

Countermeasures for Implementations of ECC

IAIK TUG

- **SSCA Countermeasures:**
  - (Re) Order the field operations in such a way that double and add/subtract look the same. Use a curve where the two operations are identical.
  - Use a multiplication algorithm with an always double and always add structure.
  - Randomize the sequence of the operations in the point multiplication algorithm (take care, probably none of the proposed algorithms provide sufficient security)
- **DSCA Countermeasures:**
  - Randomize the base point (use projective coordinates)
  - Randomize the scalar
  - Randomize the algorithm
  - Introduce random process interrupts (dummy cycles) to desynchronize the measurements.

Fault Attacks – Introduction

When an attacker has physical access to a cryptographic device, he may try to force it to malfunction.

A fault attack is an attack in which information about the message or the secret key is leaked from the output of erroneous computations.

There are several ways to introduce an error during the computation performed by the cryptographic device. Though the description of these practical means is beyond the scope of this introduction, we cite some non-invasive methods:

- spike attacks work by deviating the external power supply more than can be tolerated by the device. This will surely lead to a wrong computation.
- glitch attacks are similar to spike attacks, but target the clock contact of the integrated circuit.
- optical attacks work by focusing flash-light on the device in order to set or reset bits.

Fault Attacks – Attack Models

There are a lot of different fault attacks. Most of the time, they differ by the assumptions made about the attacker's capabilities: the way he can access and modify the memory, the power he has upon the fault occurrence time, etc. One can characterize fault attacks according to different criteria:

- control on the fault location;
- control on the fault occurrence time;
- control on the number of faulty bits induced;
- the fault model.

On the three first items, an attacker can have either no control, loose control or precise control. Fault models include: the random fault model, the bit flip model, and the bit set or reset model.

Fault Attacks – Application to ECC

IAIK TUG

The main idea behind fault attacks on implementations of ECC is:

> By disturbing the representation of a point we enforce a device to apply its point multiplication algorithm to a value which is not a point on the given but on some different curve. It is a crucial observation that the result of this computation is a point on the new probably cryptographically less strong curve which can be exploited to compute the secret key d.

Consider an EC given by $E : y^2+a_1xy+a_3y = x^3+a_2x^2+a_4x+a_6$. The parameter a_6 is not involved in the addition formula. Hence, if you input $\tilde{P} = (\tilde{x}, \tilde{y})$ with $\tilde{P} \notin E$, then the scalar multiplication $d\tilde{P}$ will take place over the curve $\tilde{E}$ with $\tilde{a}_6 = \tilde{y}^2+a_1\tilde{x}\tilde{y}+a_3\tilde{y}-\tilde{x}^3-a_2\tilde{x}^2-a_4\tilde{x}$. Assume that the point $\tilde{P}$ has been chosen so that $\tilde{E}$ is a curve with an order that has a small (or smooth) factor r and that the order of $\tilde{P}$ is equal to r. Then, the value of $d \pmod r$ can be recovered.

Hence, by repeating the attack with sufficiently many chosen points $\tilde{P}$, the value of d can be recovered by Chinese remaindering.

Fault Attacks – Attack Scenarios

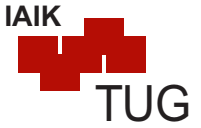
IAIK TUG

How can we induce faults such that the attack described before becomes practical?

- Software implementations: In case that a device/programm does not explicitly check whether an input point P nor the result of the computation is a point on the cryptographically strong curve, this attack is directly applicable.
- Assume that we can produce one register fault inside the device right after the point has been received as input. Then the device computes internally with a point $\tilde{P}$ which differs in exactly one bit from the input point P. By taking the output point we can determine $\tilde{a}_6$ such that $\tilde{P}$ is on a weak curve. Finding the (unknown) input point $\tilde{P}$ is simple since it differs only in one bit from the original point. Then we compute the order of the point, and if it has a small divisor r, we try to solve the DL problem.

Attacks under more general assumptions do exist . . .

(Intermediate) Conclusions



- Side-channel attacks are passive implementation attacks. Several side-channels such as timing, power, em, error messages have been exploited so far.
- Many of the implementation options which we discussed can be used to prevent those passive attacks. Mostly, this results in some penalty with respect to speed, size or interoperability.
- Fault attacks are active implementation attacks. Depending on the assumed attack model, an attacker can supply wrong parameters or even change/flip bits inside registers. There exist attacks for ECC. Countermeasures include checking the results of the computation for faults and hardware-countermeasures.

IAIK TUG

THE END

Thank you for your Attention!

Questions?

Elisabeth.Oswald@{iaik.at,esat.kuleuven.ac.be}
http://www.iaik.at
http://www.esat.kuleuven.ac.be/cosic
http://www.a-sit.at