PHILIPS

Curve based cryptography - The state of the art in smart card environments

Kim Nguyen
Cryptology Competence Center
Business Unit Identification
Philips Semiconductors GmbH

Overview

- Introduction to elliptic and hyperelliptic curves.
- Specific restraints in smart card environments.
- Example: signature algorithms based on elliptic curves.
- Some experiments with hyperelliptic curves on smartcards.
- Focus on efficient implementations.
- Secure implementations are considered in the next talk.

Elliptic curves

Cryptology Competence Center

- Consider the equation

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

  over a finite field $GF(q^n)$.
- E defines an elliptic curve over the finite field $GF(q^n)$ (certain technical conditions have to be fulfilled).
- The set $E(GF(q^n))$ of points (x,y) satisfying the equation E form an abelian group.
- The group law on $E(GF(q^n))$ can be expressed in simple algebraic formulae.

Elliptic curves

Cryptology Competence Center

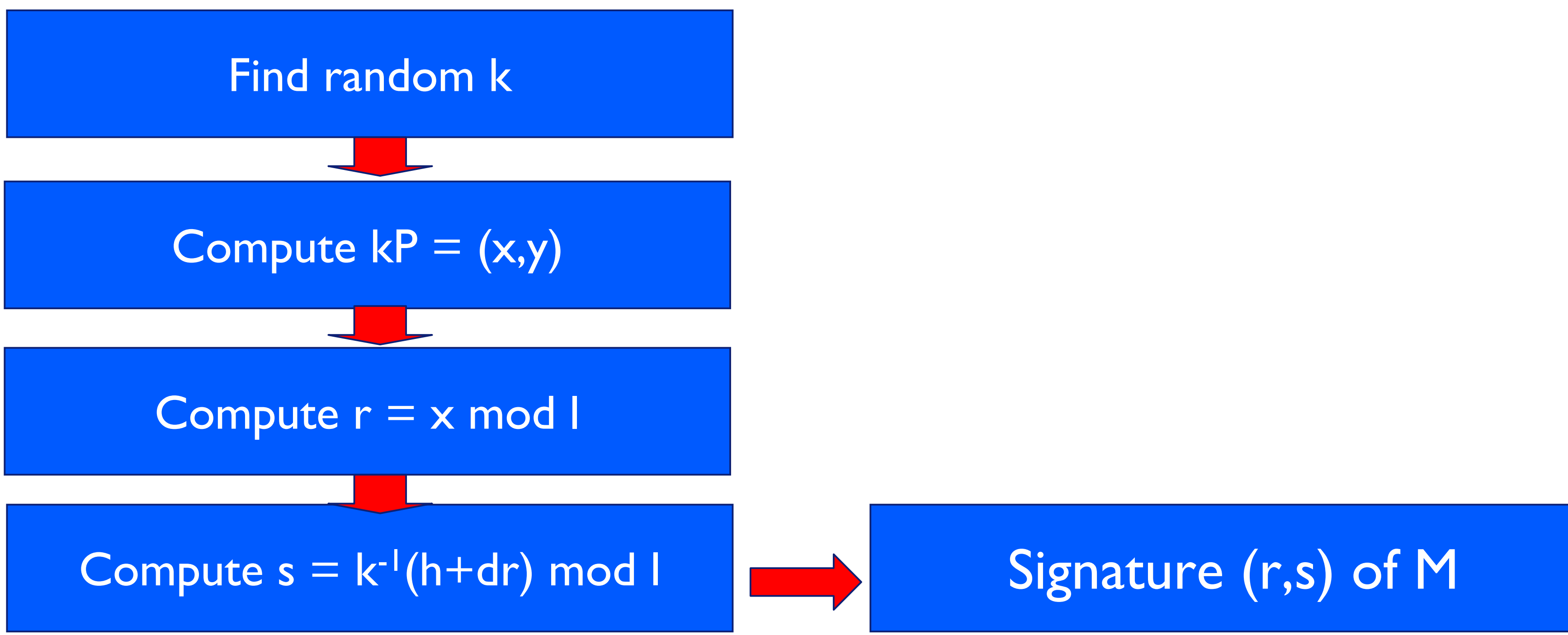
- In any abelian group we can formulate the discrete logarithm problem:
- Discrete logarithm problem in $E(GF(q^n))$: Given $P \in E(GF(q^n))$ and kP, compute k.
- Advantage: for "general" elliptic curves no subexponential attacks are known.
- Consequence:
– smaller group sizes possible (160-190 bits)
– slow increase in group size

Digital signatures based on elliptic curves

Cryptology Competence Center

ECDSA Signature Generation

- Signature generation for message M:
  private key d, hash value h=Hash(M), order l of base point P



Digital signatures based on elliptic curves

Cryptology Competence Center

ECDSA Signature Verification

- Signature verification for message M, signature (r,s), hash h: base point P, public key Q=dP, order l of base point P

Compute $w=s^{-1} \bmod l$

$u_1 = hw \bmod l, u_2 = rw \bmod l$

Compute $u_1P+u_2Q = (x,y)$

Compute $v = x \bmod l$

Check $v = r$ to verify signature

Analysis of Signature algorithms

- Two main parts:
  - scalar multiplication on elliptic curve
  - computations modulo order l of the basepoint P in order to generate signature
- Consequence:
- Modulo arithmetic is needed even if the elliptic curve is defined over $GF(2^n)$.
- Computation of modular inverses is required.
- Computation of x mod l required.
- For verification double scalar multiplication u_1P+u_2Q is needed.

ECC: Implementational choices

Scalar multiplication (recoding, precomputations, subfield curves special automorphisms etc)

Point Operations (affine/projective coordinates, point halving)

Choice of ground field (GF(p),GF(q^n),special primes,OEF etc)

ECC: Implementational choices

Cryptology Competence Center

- The choice of special curves can lead to substantial performance gains:
- Subfield curves defined over GF(q) considered over GF(q^n):
  Use of Frobenius automorphism
  can speed up scalar multiplication.
- Curves with special automorphisms:
  Similarly to the usage of the Frobenius,
  special automorphisms of a curve can considerably speed up the scalar multiplication.

ECC:
Implementational Choices

- Implementation using a dedicated arithmetic coprocessor:
  - long integer arithmetic and modular arithmetic handled by coprocessor
  - high performance
  - extra chip area
- Implementation using only a standard CPU:
  - long integer arithmetic and modular arithmetic handled by CPU
  - special field structures (e.g. optimal extension fields) or special moduli (e.g. generalized Mersenne primes) can be used to speed up the the field arithmetic considerably
  - Performance of 1- 2 s for ECDSA can be reached

ECC: Implementational Choices

- Question:
  which operations and features should be supported in hardware ?
- Absolutely necessary:

  Modular arithmetic

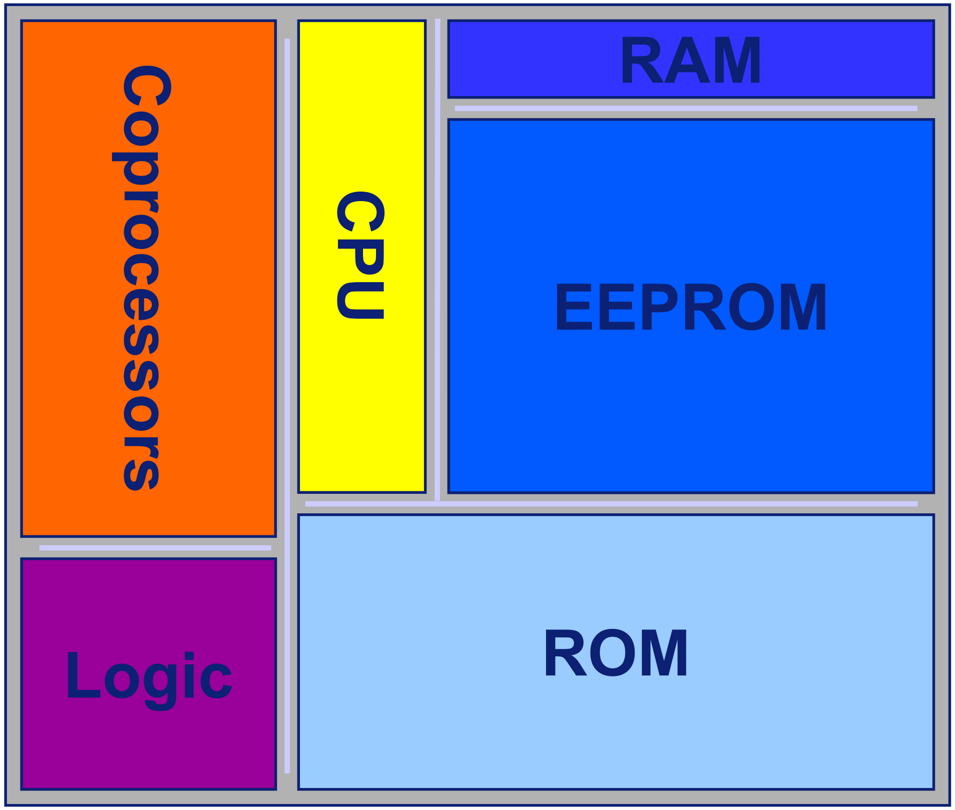
  $GF(2^n)$ arithmetic for elliptic curve calculations.
- Modular inversion is most time critical single operation.
- Support for special curves, special fields etc in hardware ?

Design philosophies

- Hardware supporting one specific type of field,
  one specific type of curve,
  even only one specific field
  or even only one curve over one specific field:
  very high performance ⇔ very low flexibility
- Flexible hardware supporting general arithmetic
  will allow flexible use of different crypto systems
  as well as easy adjustment of parameters:
  still high performance ⇔ very high flexibility

Standard **Smart Card IC Design**

Cryptology Competence Center



complete area
less than 25 mm^2

Arithmetic in Hardware

Cryptology Competence Center

- Long-integer multiplication and addition is well suited for hardware implementation.
- Modular arithmetic:
  - "traditional" modular arithmetic is not well suited for hardware implementation.
  - Reason: "school book" division with remainder is costly.
  - Much more efficient modular reduction techniques are available, which utilize computations modulo "transformed" moduli.
  - For example:
    Montgomery multiplication,
    uses the fact the reduction modulo perfect powers of two is easy.

Arithmetic in Hardware

Cryptology Competence Center

- Important topic:
  Modular inversion.
- Basically two ways to implement this:
  - Computation of x^{-1} using Fermat's Little Theorem:
    $x^{-1} \bmod p = x^{p-2} \bmod p$ (modular exponentiation)
  - Computation of x^{-1} using the Extended Euclidean Algorithm:
    $\mathrm{Gcd}(x,p)=1 \Rightarrow 1=a*p+b*x \Rightarrow 1=b*x \bmod p$.
- Modular exponentiation:
  slow, but easy to implement.
- Extended Euclidean Algorithm:
  fast, but more costly to implement.

Modular Inversion

- Ratio Modular Inversion/Modular Multiplication
  is critical for the implementation of ECC systems:
- Known values from software implementations
  (Menezes et.al. 2000)
  - GF(p): 80 to 1
  - GF(2^n): 10 to 1
- Comparison affine/projective coordinates:
  projective coordinates are preferred
  for a ratio of 10 to 1 or higher.
- Smaller ratios are possible in hardware,
  for typical smartcard coprocessors this is not feasible
  due to area and power consumption restrictions.

History of Philips Coprocessors

Cryptology Competence Center

- CORSAIR (COprocessor for RSA In a Rush) - 1991
  - optimized for 512 bit RSA
  - signature generation in less than 1 sec (512 bit)
- Fame (Fast Accelerator for Modular Exponentiation) (1995)
- FameX (eXtended) (1997)
  - optimized for 1024 bit RSA, less than 400 ms
  - flexible usage for other Public Key Crypto systems
  - scalable length of operands up to 2048 bit and higher
- FameX+GF(2^n) prototype (2000): Cooperation with Oberthur
- FameXE (ECC) (2002): part of new SmartMX platform
  - 1024 bit RSA, 100 ms
  - optimized for ECC based algorithms
  - GF(2^n), scalable length of operands

FameXE
Crypto Coprocessor

Cryptology Competence Center

- Flexible approach to arithmetic:
  based on wordwise 32bit approach
- Freely scalable computations possible.
- Hardware support for:
  - Logical operations
  - Long integer arithmetic
  - Modular arithmetic
  - $GF(2^n)$ arithmetic
  - Modular inversion

Implementation of Signature algorithms on FameXE

- Basefield GF(2^n), polynomial base.
- "General" curves
- The following timings were achieved for ECDSA signature generation using projective coordinates without precomputations (based on simulator results) :

	191 bits	157 bits
Scalar multiplication	14.7 ms	10.8 ms
Total time	15.8 ms	11.5 ms

Implementation of Signature algorithms on FameXE

- The following timings were achieved for ECDSA signature verification using projective coordinates without precomputations:

	191 bits	157 bits
Scalar multiplication	27.6 ms	24.5 ms
Total time	28.8 ms	25.6 ms

Implementation of ECDSA on HiPerSmart

Cryptology Competence Center

- HiPerSmart: Philips Semiconductors new 32 bit platform based on a SmartMIPS core
- Implementation of ECDSA based on curves over GF(2^n) without crypto coprocessor.
- Results for signature generation using projective coordinates without precomputations:

	191 bit	163 bit
Signature generation	~ 35 ms	~ 30 ms

Further topics

- Parallelism can be exploited when performing the group operations on an elliptic curve.
  This can lead to substantial speedup if the hardware supports this.
- Key generation on smart cards is an interesting topic.
  For prime fields this does not seem promising (SEA algorithm, CM method),
  in characteristic 2 much more efficient methods are available due to Satoh, Skjeerna, Gaudry, Harley and others.

Hyperelliptic curves

Cryptology Competence Center

- A hyperelliptic curve of genus g over a finite field k is given by

  C: $v^2+h(u)v=f(u)$ with f and h polynomials over k

  where:
  - $h(u)$ is of degree at most g
  - $f(u)$ is a monic polynomial of degree $2g+1$
  - certain technical conditions have to be satisfied.

The Jacobian of C

- A divisor on C is a formal sum $D=\sum m_P P$ of points P of C.
- Its degree is $\deg(D)= \sum m_P$.
- Set Div=group of all divisors on C.
- Let denote Div_0=divisors of deg zero (subgroup of Div).
- To an element f of the function field of C we associate the divisor $div(f)= \sum_{P\in C} ord_P(f)\, P$.
- A divisor D is called principal if D=div(f) for some f. P=set of all principal divisors.
- The Jacobian of C is defined by $Jac(C)= Div_0/P$.
- Jac(C) is an abelian group, hence we can base a DL system on this group.

Implications for Cryptography

Cryptology Competence Center

- Need to determine cardinality of Jac(C) (see talks by Kedlaya, Lauder, Vercauteren)
- Assume k=GF(q), and C curve of genus g.
- Then $|Jac(C)| \sim q^g$ (Weil).
- We want $|Jac(C)|$ around 2^{160}.
- Higher genus allows to go to smaller size of ground field.
- Hence for g>=5 fieldsize around 2^{32} would suffice !?
- Index Calculus tells us: only g=1,2,3 allowed.
- For genus 3 a field size of ~ 64 bits suffices.

Representation of Elements of Jac(C)

- Each element of Jac(C) can be represented by a pair of polynomials over k: (a(u),b(u)) with a(u) normalized.
- Call such an element reduced if deg(a(u))<=g.
- Main Operation is Addition of two reduced elements. This falls into two parts:

  Composition : result is semi-reduced divisor.

  Reduction : input semi-reduced, output reduced.
- Both steps can represented by operations on polynomials (Cantor algorithm).

Analysis of Cantor Algorithm

- Need polynomial arithmetic over finite field k:
  - Gcd(a,b): Greatest common divisor.
  - Div/Mod: Polynomial division with remainder.
- These are complicated algorithms.
- Very efficient field arithmetic is needed (esp. inversion!).
- Efficiency of complete implementation in hardware is unclear (see Th. Wollingers M.Sc. thesis, WPI 2001).
- Active research on optimal hardware environment done by Philips Semiconductors in EU IST project AREHCC.

Explicit formulae

Cryptology Competence Center

- For genus two we have explicit formulae which can replace the general Cantor algorithm:
- Spallek (Ph.D. thesis, Essen 1996)
- Harley
- Takahashi (SCIS 2002)
- Miyamoto,Doi,Matsuo,Chao,Tsuji (SCIS 2002)
- Boston, Clancy (CHES 2002)
- Lange (Preprint, 9-2002)
- For genus three first results were obtained by Pelzl, Paar (Uni Bochum).

Explicit formulae

Cryptology Competence Center

- Especially interesting for smartcard applications: We have "affine" and "projective" formulae
  - affine meaning one inversion for each addition or doubling step is used
  - projective meaning one inversion is used at the end of the scalar multiplication
- Idea behind "projective" computations:

  use representation of divisor with non-normalized polynomials Normalize at the very end of scalar multiplication.
- This idea can also be applied to the Cantor algorithm itself (Diploma thesis U. Krieger, 1997)

Implementation of HEC based systems on FameXE

Cryptology Competence Center

- Using the projective formulae by Lange (17-09-2002) the following timings were achieved (based on simulator results):

  Jacobian of a general hyperelliptic curve of genus 2 over a field $GF(2^n)$ of ~ 90 bits

  Scalar multiplication k*D using Double-and-Add without precomputations on FameXE:

Field size	90 bit
Scalar multiplication	~ 30 ms

Thank you for your attention !

PHILIPS