Bern University of Applied Sciences

UniCrypt 2.0

Mathematical and Cryptographic Concepts

Rolf Haenni

E-Voting Seminar, Biel (Switzerland), March 25th, 2014

Inhalt

Introduction

Motivation

- Multiple e-voting projects since 2008
- Various protocols implemented [CGS97, JCJ05, . . .]
- Cryptographic primitives re-implemented
  - Secret-sharing
  - Pedersen commitments
  - ElGamal encryption and re-encryption
  - Zero-knowledge proofs
  - Cryptographic mixing
  - Elliptic curves

    . . .
- No suitable library available off the shelf

Introductory Example: JCA

```java
KeyGenerator keyGenerator =
    KeyGenerator.getInstance("AES");
keyGenerator.init(128);
SecretKey key = keyGenerator.generateKey();

Cipher cipher =
    Cipher.getInstance("AES/ECB/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, key);

byte[] message = new Random().getBytes(new byte[20]);
byte[] encrypted = cipher.doFinal(message);

cipher.init(Cipher.DECRYPT_MODE, key);
byte[] decrypted = cipher.doFinal(encrypted);
```

Introductory Example: UniCrypt

```
AESEncryptionScheme aes =
    AESEncryptionScheme.getInstance();

Element key = aes.generateKey();

Element message =
    aes.getMessageSpace().getRandomElement(20);

Element encryption = aes.encrypt(key, message);

Element decryption = aes.decrypt(key, encryption);
```

Project Milestones

- February'12: UniVote project launched
- July'12: First inofficial UniCrypt release (student project)
- February'13: Second inofficial release (part of UniVote)
- March – June'13: Multiple elections in Berne, Zürich, Lucerne
- August'13: Independent project on GitHub
- September'13 – February'14: Complete re-design
- December'13: Proof of shuffle implemented (Wikström)
- February'14: Alpha version used in MobiVote
- February'14: First public talk at TU Darmstadt

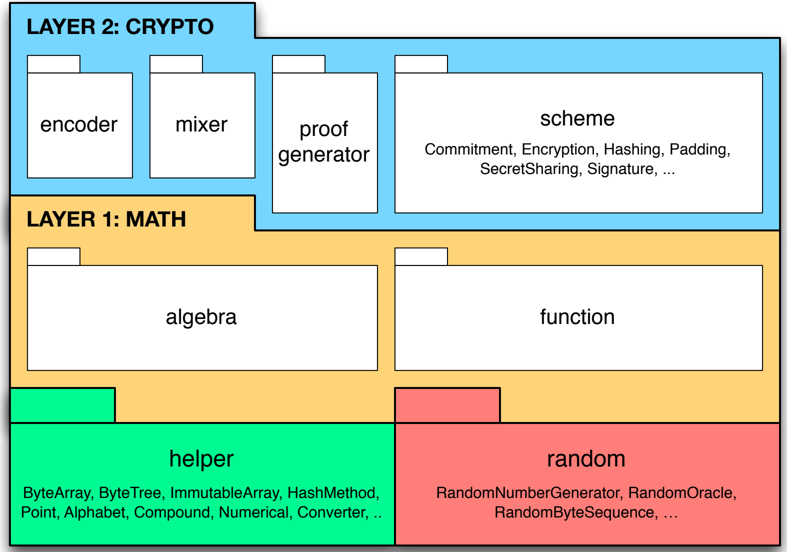
Bern University
of Applied Sciences

Design Principles and Architecture

Design Principles

- Full coherence with mathematical and cryptographic concepts
- Consistent and self-explanatory nomenclature
- Clean and intuitive APIs
- Convenience methods for improved easy of use
- Generic types (hidden from the developer if possible)
- Consistent coding style
- Immutable objects only
- Design patterns (if useful)
- Memoizing (if useful)
- No cryptographic black-boxes (e.g. random generator)
- Java 6 compatibility (Android)

Architecture



Conventions

- Constructors are private or protected (no tests, only field initializations)
- Fields are private or protected and final
- Object creation by static factory methods (perform all tests)
  - `getInstance(...);`
  - `getRandomInstance(...);`
- Every interface has a corresponding abstract class (e.g. `Set` and `AbstractSet`)
- Abstract classes implement every method as far as possible, which then calls
  - `defaultMethodName(...);` $\rightarrow$ can be overridden
  - `abstractMethodName(...);` $\rightarrow$ needs to be overridden
- Value `null` is never allowed

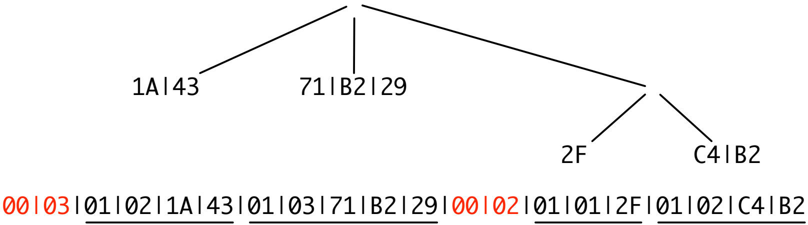
Helper Classes

ByteArray and ImmutableArray

- Problems of `byte[]` and `Object[]`:
  - Arrays are no classes: no additional functionality
  - Mutable: cloning necessary to avoid side effects
  - Extracting sub-arrays: $O(s)$ time
  - Uniform array: $O(n)$ space
- Advantages of `ByteArray` and `ImmutableArray<T>`
  - Added functionality: and, or, xor, not, append, extract, ...
  - Immutable: no side effects
  - Extracting operation: $O(1)$ time
  - Uniform array: $O(1)$ space
  - Proper `toString()`

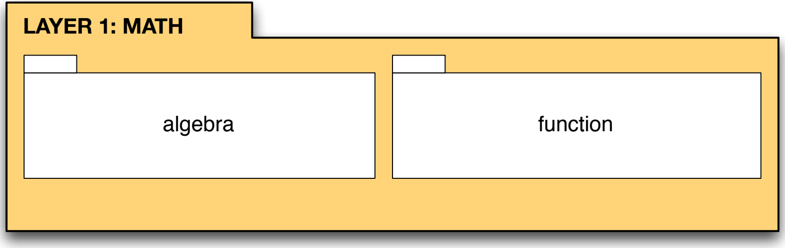
ByteTree

- A byte tree is a tree with byte arrays attached to its leafs (D. Wikström)
- `ByteTree` is an immutable implementation of byte trees, which implements the conversion to `ByteArray` and back according to Wikström
- Simplified example:



Layer 1: Mathematics

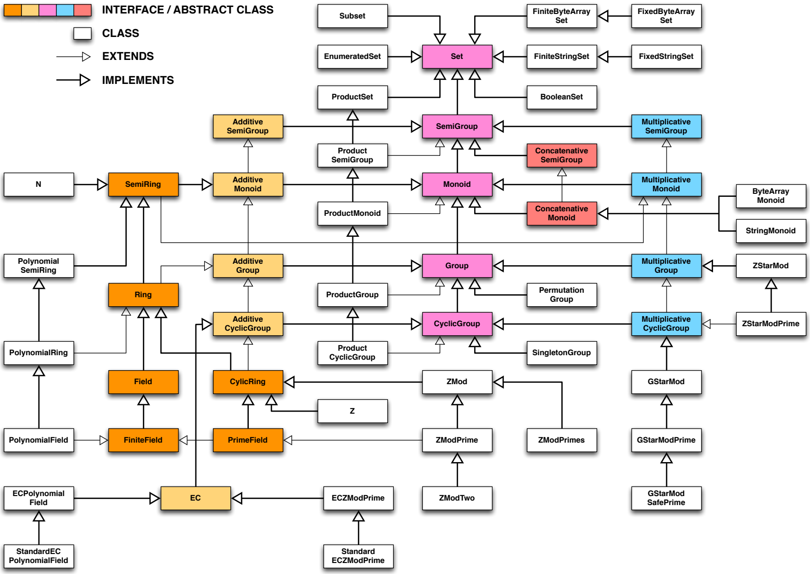
Architecture: Layer 1



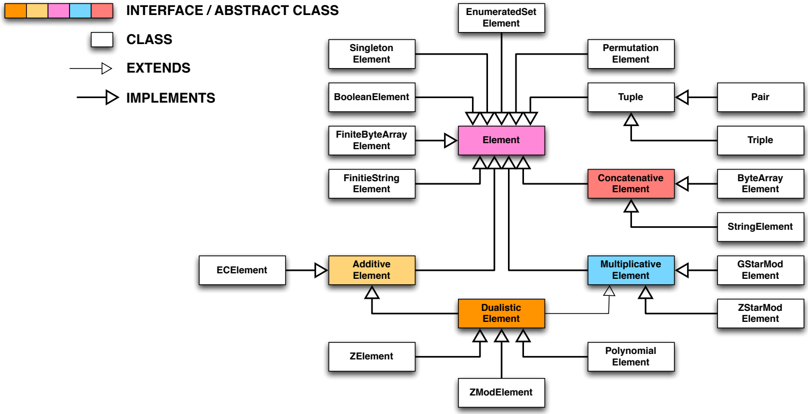
Abstract Algebra

- Set S
- Semigroup $(S, \circ)$
- Monoid $(S, \circ, id)$
- Group $(S, \circ, id, Inv)$
- Cyclic group $(S, \circ, id, Inv, g)$
- Semiring $(S, +, \times, 0, 1)$
  $=$ Monoid $(S, +, 0)$ & Monoid $(S, \times, 1)$
- Ring $(S, +, \times, 0, 1, -)$
  $=$ Group $(S, +, 0, -)$ & Monoid $(S, \times, 1)$
- Field $(S, +, \times, 0, 1, -, ^{-1})$
  $=$ Group $(S, +, 0, -)$ & Group $(S \setminus 0, \times, 1, ^{-1})$
- Finite field $|S| = p^k$, prime field $|S| = p$

Algebra Package



Algebra Package



Sets

- The interface `Set` has a generic type `V` (see next slide)
- Operations:
  - `BigInteger getOrder()`
  - `Element<V> getElement(V value)`
  - `Element<V> getRandomElement()`
  - `Boolean contains(Element elt)`
- Examples:
  - `BooleanSet`: $B = \{true, false\}$
  - `EnumeratedSet`: $S = \{s_1, \ldots, s_n\}$
  - `FixedStringSet`: $S_n = \mathcal{A}^n$
  - `FiniteStringSet`: $S_{m,n} = \mathcal{A}^m \cup \cdots \cup \mathcal{A}^n$
  - `FixedByteArraySet`: $B_n = [0, 255]^n$
  - `FiniteByteArraySet`: $B_{m,n} = [0, 255]^m \cup \cdots \cup [0, 255]^n$

Elements

- Every element ...
  - belongs to a set
  - has a value for storing its information (generic type `V`)
  - can be converted to a positive integer (and back)
  - can be converted to a byte array or byte tree (and back)
  - can be hashed
- Methods:
  - `Set<V> getSet()`
  - `V getValue()`
  - `BigInteger getBigInteger()`
  - `ByteArray getByteArray()`
  - `ByteTree getByteTree()`
  - `ByteArray getHashValue()`
- Examples:
  - `BooleanElement`, `EnumeratedSetElement`, `FiniteStringElement`, `FiniteByteArrayElement`

Semigroups and Monoids

- Methods:
  - `Element<V> apply(Element elt1, Element elt2)`
  - `Element<V> selfApply(Element elt, BigInteger n)`
  - `Element<V> getIdentityElement()`
  - `boolean isIdentityElement(Element elt)`
- Corresponding methods exist for semigroup and monoid elements
- Examples:
  - `StringMonoid`: $S_b = \mathcal{A}^0 \cup \mathcal{A}^b \cup \mathcal{A}^{2b} \cup \cdots$
  - `ByteArrayMonoid`: $B_b = [0,255]^0 \cup [0,255]^b \cup [0,255]^{2b} \cup \cdots$

Groups and Cyclic Groups

- Methods:
  - `Element<V> invert(Element elt)`
  - `Element<V> getDefaultGenerator()`
  - `Element<V> getRandomGenerator()`
  - `Element<V> getIndependentGenerator(int index)`
  - `boolean isGenerator(Element elt)`
- Corresponding methods exist for group elements
- Examples:
  - `PermutationGroup`: $\Pi_n = \{\pi : \text{ permutation of size } n\}$
  - `ZStarMod`: $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n : gcd(x, n) = 1\}$
  - `ZStarModPrime`: $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$
  - `GStarMod`: $\mathbb{G}_q \subset \mathbb{Z}_n^*$ (cyclic subgroup of prime order q)
  - `GStarModPrime`: $\mathbb{G}_q \subset \mathbb{Z}_p^*$
  - `GStarModSafePrime`: $\mathbb{G}_q \subset \mathbb{Z}_p^*$ for $p = 2q + 1$
  - `ECZModPrime`: $E(\mathbb{Z}_p) = \{(x, y) : x^2 = y^3 + ax + b\} \cup \{\infty\}$
  - `ECPolynomialField`: $E(\mathbb{Z}_{2^p})$

Additive Algebraic Structures

- In some cases, the operator is written additively:
  - `AdditiveSemiGroup`, `AdditiveMonoid`, `AdditiveGroup`, `AdditiveCyclicGroup`
  - `AdditiveElement`
- Methods (return type `AdditiveElement<V>`):
  - `add(Element elt1, Element elt2)`
  - `times(Element elt, BigInteger n);`
  - `getZeroElement()`
  - `isZeroElement(Element elt)`
  - `negate(Element element)`
  - `subtract(Element elt1, Element elt2);`
- Examples:
  - `ECZModPrime`, `ECPolynomialField`

Multiplicative Algebraic Structures

- In some cases, the operator is written multiplicatively:
  - `MultiplicativeSemiGroup`, `MultiplicativeMonoid`, `MultiplicativeGroup`, `MultiplicativeCyclicGroup`
  - `MultiplicativeElement`
- Methods (return type `MultiplicativeElement<V>`):
  - `multiply(Element elt1, Element elt2)`
  - `power(Element elt, BigInteger n);`
  - `getOneElement()`
  - `isOneElement(Element elt)`
  - `oneOver(Element element)`
  - `divide(Element elt1, Element elt2);`
- Examples:
  - `ZStarMod`, `ZStarModPrime`, `GStarMod`, `GStarModPrime`, `GStarModSafePrime`

Concatenative Algebraic Structures

- In some cases, the operator is written concatenatively:
  - `ConcatenativeSemiGroup`, `ConcatenativeMonoid`
  - `ConcatenativeElement`
- Methods (return type `ConcatenativeElement<V>`):
  - `concatenate(Element elt1, Element elt2)`
  - `selfConcatenate(Element elt, BigInteger n);`
  - `getEmptyElement()`
  - `isEmptyElement(Element elt)`
- Examples:
  - `ByteArrayMonoid`, `ByteArrayMonoid`

Semirings, Rings, Fields

- Methods (return type `DualisticElement<V>`):
  - `SemiRing` inherits all methods from `AdditiveMonoid` and `MultiplicativeMonoid`
  - `Ring` inherits additional methods from `AdditiveGroup`
  - Convention: `add`=`apply` and `times`=`selfApply`
- Examples:
  - `N`: Semiring of natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$
  - `Z`: Ring of integers $\mathbb{Z} = \{0, \pm 1, \pm 2, \ldots\}$
  - `ZMod`: Ring $\mathbb{Z}_n$ of integers modulo n (residue classes)
  - `ZModPrime`: Prime field $\mathbb{Z}_p$
  - `PolynomialSemiRing`: Polynomial semiring $S[x]$ over ring S
  - `PolynomialRing`: Polynomial ring $R[x]$ over ring R
  - `PolynomialField`: Polynomial field $F[x]$ over field F

Cartesian Products and Tuples

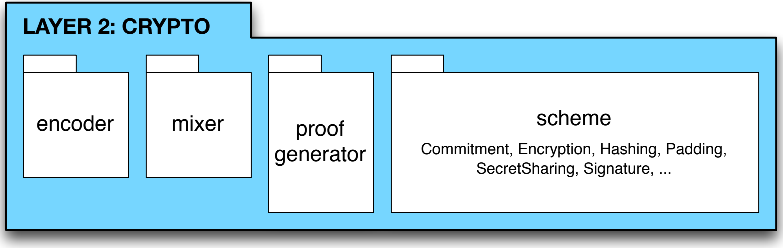
- Sets and elements can be composed recursively
- Cartesian products $S_1 \times \cdots \times S_n$
  - `ProductSet`
  - `ProductSemiGroup`
  - `ProductMonoid`
  - `ProductGroup`
  - `ProductCyclicGroup`
- Corresponding combined elements
  - `Tuple`: general tuples $(e_1, \ldots, e_n) \in S_1 \times \cdots \times S_n$
  - `Pair`: tuples $(e_1, e_2) \in S_1 \times S_2$ of arity 2
  - `Triple`: tuples $(e_1, e_2, e_3) \in S_1 \times S_2 \times S_3$ of arity 3
- Methods:
  - `int getArity()`
  - `Set getAt(int index)`
  - `Element getAt(int index)`

Functions

- `Function`: mathematical concept of a function $f : X \rightarrow Y$
  - `public Set getDomain();`
  - `public Set getCoDomain();`
  - `public Element apply(Element elt);`
- There is a large set of predefined functions
  - `AdapterFunction`, `AdditionFunction`, `ConstantFunction`, `ConvertFunction`, `EqualityFunction`, `HashFunction`, `IdentityFunction`, `InvertFunction`, `ModuloFunction`, `MultiplicationFunction`, `PermutationFunction`, `PowerFunction`, `SelectionFunction`, ...
- Functions can be combined in two ways
  - `ComposedFunction`: $f(x) = f_1 \circ \cdots \circ f_n(x) = f_1(f_2(\cdots f_n(x)))$
  - `ProductFunction`: $f(x_1, \ldots, x_n) = (f_1(x_1), \ldots, f_n(x_n))$

Layer 2: Cryptography

Architecture: Layer 2



Cryptographic Schemes

- Berry Schoenmakers
  - "A *cryptographic algorithm* is a transformation, which on a given input value produces an output value, achieving certain security objectives"
  - "A *cryptographic scheme* is a suite of related cryptographic algorithms achieving certain security objectives"
- UniCrypt support various cryptographic schemes
  - `CommitmentScheme`: `commit(...)`, `decommit(...)`
  - `EncryptionScheme`: `encrypt(...)`, `decrypt(...)`
  - `HashingScheme`: `hash(...)`, `check(...)`
  - `PaddingScheme`: `pad(...)`, `unpad(...)`
  - `SecretSharingScheme`: `share(...)`, `recover(...)`
  - `SignatureScheme`: `sign(...)`, `verify(...)`
- In UniCrypt, the input value of a scheme is called *message*
  - `Scheme` has a single method `Set getMessageSpace()`

Shamir Secret Sharing

- Prime field $\mathbb{Z}_p$
- Secret $s \in \mathbb{Z}_p$ to share among n people
- Threshold $t \leq n$
- Polynomial $f(x) = s + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}$ for $a_i \in \mathbb{Z}_p$
- Share $s_i = (x_i, f(x_i))$, for $x_i \in \mathbb{Z}_p$ and $i = 1, \ldots, n$
- Recovering of s using Lagrange interpolation

ElGamal Encryption

- Cyclic prime order subgroup $G_q \subset Z_p^*$ for $p = 2q + 1$
- Generator $g \in G_q$
- Private key $x \in \mathbb{Z}_q$
- Public key $y = g^x \in G_q$
- Randomization $r \in \mathbb{Z}_q$
- Message $m \in G_q$
- Encryption: $Enc_y(m, r) = (g^r, m \cdot y^r) \in G_q \times G_q$
- Decryption: $Dec_x(a, b) = b/a^x$

Encoders

- An encoder represents an injective mapping between two sets
  - `public Element encode(Element elt)`
  - `public Element decode(Element elt)`

- Example: Encrypt lower-case string with ElGamal
  - Recall that $m \in G_q \subset \mathbb{Z}_p^*$
  - For example $G_{11} = \{1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18\} \subset \mathbb{Z}_{23}^*$
  - We need $f : S_n \rightarrow G_q$ and $f^{-1} : G_q \rightarrow S_n$,
  - We can construct f as a composed function $f = f_1 \circ f_2$ for
    $f_1 : S_n \rightarrow \mathbb{Z}_q$ (`FiniteStringToZModEncoder`)
    $f_2 : \mathbb{Z}_q \rightarrow G_q$ (`ZModToGStarModSafePrimeEncoder`)

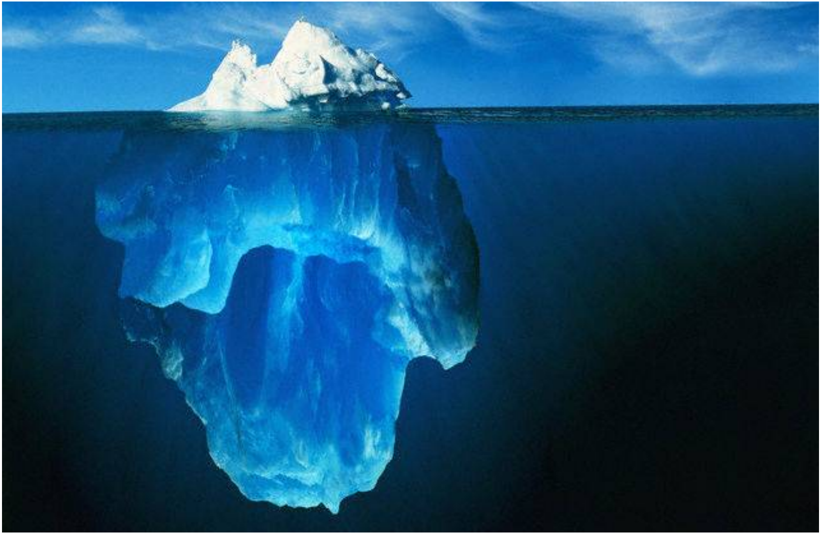
Summary and Outlook

Summary

- UniCrypt = Java library with advanced mathematical and cryptographic primitives
- Offers clean and intuitive APIs
- Growing in size
  - 68 interfaces
  - 217 classes
  - 34553 lines of codes (incl. comments, excl. tests)
- Open-source: abailable on GitHub
- Free for academic or non-commercial usage (dual license)
- Collaborations are welcome

Outlook

- Stage of development: alpha
- Important components under development
  - elliptic curves
  - true random generators
- Important components missing
  - signature schemes
  - certificates
  - further encryption schemes (Paillier, etc.)
  - further types of zero-knowledge proofs
  - other cryptographic schemes
- Improper exception handling (proper concept missing)
- Documentation largely missing
- Insufficient code coverage by existing JUnit tests



Source: http://www.djibang.org/wp/wp-content/uploads/eisberg.jpg

Questions?

http://e-voting.bfh.ch

https://github.com/bfh-evg/unicrypt