Development of security-critical software with Spark/Ada at secunet

Stefan Berghofer

30.5.2017
Agenda

1. Introduction

2. A Link between Why3 and Isabelle

3. Applications

4. Conclusion
1. Introduction

2. A Link between Why3 and Isabelle

3. Applications

4. Conclusion
About secunet Security Networks AG

One of Germany’s leading providers of IT security
Security partner of the Federal Republic of Germany
More than 400 employees
Major shareholder is Giesecke & Devrient, Munich

Business Units:
- Public sector
  - Public Authorities, Homeland Security, Defence
- Business sector
  - Automotive, Critical Infrastructures

More details: www.secunet.com
About secunet Security Networks AG

- One of Germany’s leading providers of IT security
About secunet Security Networks AG

- One of Germany’s leading providers of IT security
- Security partner of the Federal Republic of Germany
About secunet Security Networks AG

- One of Germany’s leading providers of IT security
- Security partner of the Federal Republic of Germany
- More than 400 employees
About secunet Security Networks AG

- One of Germany’s leading providers of IT security
- Security partner of the Federal Republic of Germany
- More than 400 employees
- Major shareholder is Giesecke & Devrient, Munich
About secunet Security Networks AG

- One of Germany’s leading providers of IT security
- Security partner of the Federal Republic of Germany
- More than 400 employees
- Major shareholder is Giesecke & Devrient, Munich
- Business Units:

  - Public sector
    - Public Authorities, Homeland Security, Defence
  - Business sector
    - Automotive, Critical Infrastructures

More details: www.secunet.com
About secunet Security Networks AG

- One of Germany’s leading providers of IT security
- Security partner of the Federal Republic of Germany
- More than 400 employees
- Major shareholder is Giesecke & Devrient, Munich
- Business Units:
  - Public sector
About secunet Security Networks AG

- One of Germany’s leading providers of IT security
- Security partner of the Federal Republic of Germany
- More than 400 employees
- Major shareholder is Giesecke & Devrient, Munich
- Business Units:
  - **Public sector**
    - Public Authorities, Homeland Security, Defence
About secunet Security Networks AG

- One of Germany’s leading providers of IT security
- Security partner of the Federal Republic of Germany
- More than 400 employees
- Major shareholder is Giesecke & Devrient, Munich
- Business Units:
  - Public sector
    - Public Authorities, Homeland Security, Defence
  - Business sector
About secunet Security Networks AG

- One of Germany’s leading providers of IT security
- Security partner of the Federal Republic of Germany
- More than 400 employees
- Major shareholder is Giesecke & Devrient, Munich
- Business Units:
  - Public sector
    - Public Authorities, Homeland Security, Defence
  - Business sector
    - Automotive, Critical Infrastructures
About secunet Security Networks AG

- One of Germany’s leading providers of IT security
- Security partner of the Federal Republic of Germany
- More than 400 employees
- Major shareholder is Giesecke & Devrient, Munich
- Business Units:
  - Public sector
    - Public Authorities, Homeland Security, Defence
  - Business sector
    - Automotive, Critical Infrastructures
- More details: www.secunet.com
Context

High-security VPN gateways and clients
Context

High-security VPN gateways and clients

- Process various categories of data with different classification
Context

High-security VPN gateways and clients

- Process various categories of data with different classification
- Prevent unintended information flow between domains
Context

High-security VPN gateways and clients

- Process various categories of data with different classification
- Prevent unintended information flow between domains

Development approach
Context

High-security VPN gateways and clients

- Process various categories of data with different classification
- Prevent unintended information flow between domains

Development approach

- Design goal: Keep trusted computing base small
Context

High-security VPN gateways and clients

- Process various categories of data with different classification
- Prevent unintended information flow between domains

Development approach

- Design goal: Keep trusted computing base small
- Component-based architecture
Context

High-security VPN gateways and clients

- Process various categories of data with different classification
- Prevent unintended information flow between domains

Development approach

- Design goal: Keep trusted computing base small
- Component-based architecture
  - Few small, trusted components, e.g. encryption / decryption
High-security VPN gateways and clients

- Process various categories of data with different classification
- Prevent unintended information flow between domains

Development approach

- Design goal: Keep trusted computing base small
- Component-based architecture
  - Few small, trusted components, e.g. encryption / decryption
  - Larger untrusted components, e.g. user sessions, device drivers, network protocol stack, ...
Context

**High-security VPN gateways and clients**

- Process various categories of data with different classification
- Prevent unintended information flow between domains

**Development approach**

- Design goal: Keep trusted computing base small
- Component-based architecture
  - Few small, trusted components, e.g. encryption / decryption
  - Larger untrusted components, e.g. user sessions, device drivers, network protocol stack, ...  
- **Separation kernel** controls interaction of components
Context

High-security VPN gateways and clients

- Process various categories of data with different classification
- Prevent unintended information flow between domains

Development approach

- Design goal: Keep trusted computing base small
- Component-based architecture
  - Few small, trusted components, e.g. encryption / decryption
  - Larger untrusted components, e.g. user sessions, device drivers, network protocol stack, ... 
- Separation kernel controls interaction of components
- Implement / verify trusted components using SPARK
Context

High-security VPN gateways and clients
- Process various categories of data with different classification
- Prevent unintended information flow between domains

Development approach
- Design goal: Keep trusted computing base small
- Component-based architecture
  - Few small, trusted components, e.g. encryption / decryption
  - Larger untrusted components, e.g. user sessions, device drivers, network protocol stack, ... 
- Separation kernel controls interaction of components
- Implement / verify trusted components using SPARK
- Prove at least absence of runtime exceptions for all trusted components, for some also functional correctness
2010 Implementation of components in Spark 2005
Spark at secunet

2010  Implementation of components in Spark 2005

2011  Verification of components using Spark and Isabelle
Spark at secunet

2010  Implementation of components in Spark 2005

2011  Verification of components using Spark and Isabelle

2013  Isabelle driver for Why3
SPARK at secunet

2010  Implementation of components in SPARK 2005
2011  Verification of components using SPARK and Isabelle
2013  Isabelle driver for Why3
2014  Switch to SPARK 2014
Spark at secunet

2010 Implementation of components in Spark 2005
2011 Verification of components using Spark and Isabelle
2013 Isabelle driver for Why3
2014 Switch to Spark 2014
   Introduction of Muen separation kernel
SPARK at secunet

2010  Implementation of components in SPARK 2005

2011  Verification of components using SPARK and Isabelle

2013  Isabelle driver for Why3

2014  Switch to SPARK 2014
       Introduction of Muen separation kernel

2017  Proof of properties of Muen separation kernel
       (ongoing)
Component-Based Architecture

Red Linux

Enc

Dec

Key Manager

Internet

Black Linux

Key Agreement (Diffie-Hellman)

Authentication (X509 Certificates)
Component-Based Architecture

Red Linux

Dec

Key Manager

Enc

{P}_K

{P'}_{K'}

K

K'

P

P'

Internet

Black Linux

Enc

Dec

Key Agreement (Diffie-Hellman)

Authentication (X509 Certificates)
Component-Based Architecture

Key Manager

Red Linux

Dec

Enc

Key Agreement (Diffie Hellman)

Internet

Black Linux
Component-Based Architecture

Red Linux

Dec

Key Manager

Enc

Black Linux

Internet

Key Agreement
(Diffie Hellman)

Authentication
(X509 Certificates)
Verification Approaches

- **Automatic**
  - Auto-Active
  - Use lemma subprograms to help automatic provers

- **Interactive**
  - Using Coq or Isabelle

Why interactive verification?

- About 10% – 40% of the VCs generated from our codebase are not proved automatically by SMT solvers
- Lemma subprograms difficult to synthesize for complex proofs (tool support?)
- Complex external specifications can be linked to Spark code using ghost functions
Verification Approaches

- Automatic

- Interactive using Coq or Isabelle

Why interactive verification?

About 10% – 40% of the VCs generated from our codebase are not proved automatically by SMT solvers.

Lemma subprograms difficult to synthesize for complex proofs (tool support?)

Complex external specifications can be linked to Spark code using ghost functions.
Verification Approaches

- Automatic
- Auto-Active
Verification Approaches

- Automatic
- Auto-Active
  use lemma subprograms to help automatic provers

Why interactive verification?
About 10% – 40% of the VCs generated from our codebase are not proved automatically by SMT solvers.
Lemma subprograms diicult to synthesize for complex proofs (tool support?)
Complex external specifications can be linked to Spark code using ghost functions.
Verification Approaches

- Automatic
- Auto-Active
  - use lemma subprograms to help automatic provers
- Interactive

About 10% – 40% of the VCs generated from our codebase are not proved automatically by SMT solvers. Lemma subprograms difficult to synthesize for complex proofs (tool support?)

Complex external specifications can be linked to Spark code using ghost functions.
Verification Approaches

- Automatic
- Auto-Active
  use lemma subprograms to help automatic provers
- Interactive
  using Coq or Isabelle
Verification Approaches

- Automatic
- Auto-Active
  - use lemma subprograms to help automatic provers
- Interactive
  - using Coq or Isabelle

**Why interactive verification?**
Verification Approaches

- Automatic
- Auto-Active
  use lemma subprograms to help automatic provers
- Interactive
  using Coq or Isabelle

Why interactive verification?

- About 10% – 40% of the VCs generated from our codebase are not proved automatically by SMT solvers
Verification Approaches

- Automatic
- Auto-Active
  use lemma subprograms to help automatic provers
- Interactive
  using Coq or Isabelle

Why interactive verification?

- About 10% – 40% of the VCs generated from our codebase are not proved automatically by SMT solvers
- Lemma subprograms difficult to synthesize for complex proofs (tool support?)
Verification Approaches

- Automatic
- Auto-Active
  use lemma subprograms to help automatic provers
- Interactive
  using Coq or Isabelle

Why interactive verification?

- About 10% – 40% of the VCs generated from our codebase are not proved automatically by SMT solvers
- Lemma subprograms difficult to synthesize for complex proofs (tool support?)
- Complex external specifications can be linked to Spark code using ghost functions
1. Introduction

2. A Link between Why3 and Isabelle

3. Applications

4. Conclusion
Isabelle/HOL

Interactive theorem prover
Developed (since 1986) by Prof. Larry Paulson (Cambridge), Prof. Tobias Nipkow (Munich) and Dr. Markus Wenzel
Widely used, e.g. in the L4 verified project at NICTA
Specifications written in a functional programming language with logical operators
Design philosophy
I Inferences may only be performed by small kernel "LCF approach" [Robin Milner]
I Definitional theory extension
New concepts must be introduced using already existing and more primitive concepts.

More information: isabelle.in.tum.de

Page 10

30.5.2017

Development of security-critical software
Interactive theorem prover
- Interactive theorem prover
- Developed (since 1986) by Prof. Larry Paulson (Cambridge), Prof. Tobias Nipkow (Munich) and Dr. Markus Wenzel
Isabelle/HOL

- Interactive theorem prover
- Developed (since 1986) by Prof. Larry Paulson (Cambridge), Prof. Tobias Nipkow (Munich) and Dr. Markus Wenzel
- Widely used, e.g. in the L4 verified project at NICTA
Isabelle/HOL

- Interactive theorem prover
- Developed (since 1986) by Prof. Larry Paulson (Cambridge), Prof. Tobias Nipkow (Munich) and Dr. Markus Wenzel
- Widely used, e.g. in the L4 verified project at NICTA
- Specifications written in a functional programming language with logical operators
Interactive theorem prover

Developed (since 1986) by Prof. Larry Paulson (Cambridge), Prof. Tobias Nipkow (Munich) and Dr. Markus Wenzel

Widely used, e.g. in the L4 verified project at NICTA

Specifications written in a functional programming language with logical operators

Design philosophy
Interactive theorem prover

Developed (since 1986) by Prof. Larry Paulson (Cambridge), Prof. Tobias Nipkow (Munich) and Dr. Markus Wenzel

Widely used, e.g. in the L4 verified project at NICTA

Specifications written in a functional programming language with logical operators

Design philosophy

- Inferences may only be performed by small kernel
  “LCF approach” [Robin Milner]
Isabelle/HOL

- Interactive theorem prover
- Developed (since 1986) by Prof. Larry Paulson (Cambridge), Prof. Tobias Nipkow (Munich) and Dr. Markus Wenzel
- Widely used, e.g. in the L4 verified project at NICTA
- Specifications written in a functional programming language with logical operators
- Design philosophy
  - Inferences may only be performed by small kernel “LCF approach” [Robin Milner]
  - Definitional theory extension
    New concepts must be introduced using already existing and more primitive concepts.
Isabelle/HOL

- Interactive theorem prover
- Developed (since 1986) by Prof. Larry Paulson (Cambridge), Prof. Tobias Nipkow (Munich) and Dr. Markus Wenzel
- Widely used, e.g. in the L4 verified project at NICTA
- Specifications written in a functional programming language with logical operators
- Design philosophy
  - Inferences may only be performed by small kernel “LCF approach” [Robin Milner]
  - Definitional theory extension
    New concepts must be introduced using already existing and more primitive concepts.
- More information: isabelle.in.tum.de
Isabelle Driver for Why3

- Inspired by Coq driver
Isabelle Driver for Why3

- Inspired by Coq driver
- Uses data exchange format that is easy to generate and parse
Isabelle Driver for Why3

- Inspired by Coq driver
- Uses data exchange format that is easy to generate and parse (XML, no Isabelle theory files)
Isabelle Driver for Why3

- Inspired by Coq driver
- Uses data exchange format that is easy to generate and parse (XML, no Isabelle theory files)
- Consists of two parts
Isabelle Driver for Why3

- Inspired by Coq driver
- Uses data exchange format that is easy to generate and parse (XML, no Isabelle theory files)
- Consists of two parts
  - Why3 Generates XML file with definitions and VCs

...
Isabelle Driver for Why3

- Inspired by Coq driver
- Uses data exchange format that is easy to generate and parse (XML, no Isabelle theory files)
- Consists of two parts
  - Why3 Generates XML file with definitions and VCs
  - Isabelle Parses XML file, performs definitions, manages VCs
Isabelle Driver for Why3

- Inspired by Coq driver
- Uses data exchange format that is easy to generate and parse (XML, no Isabelle theory files)
- Consists of two parts
  - **Why3** Generates XML file with definitions and VCs
  - **Isabelle** Parses XML file, performs definitions, manages VCs
- Strict separation between generated and user-edited content
Isabelle Driver for Why3

- Inspired by Coq driver
- Uses data exchange format that is easy to generate and parse (XML, no Isabelle theory files)
- Consists of two parts
  - Why3 Generates XML file with definitions and VCs
  - Isabelle Parses XML file, performs definitions, manages VCs
- Strict separation between generated and user-edited content
- No fragile heuristics for parsing and interpreting edited parts
Isabelle Driver for Why3

- Inspired by Coq driver
- Uses data exchange format that is easy to generate and parse (XML, no Isabelle theory files)
- Consists of two parts
  - Why3 Generates XML file with definitions and VCs
  - Isabelle Parses XML file, performs definitions, manages VCs
- Strict separation between generated and user-edited content
- No fragile heuristics for parsing and interpreting edited parts
- System keeps track of proved / unproved VCs
Spark Toolchain

Source Files
(.*.ads,.*.adb)
**SPARK Toolchain**

Source Files

\((*.ads, *.adb)\) → gnatprove
**SPARK Toolchain**

Source Files (*.ads, *.adb) → gnatprove → Why3 Files (*.mlw)
**SPARK Toolchain**

Source Files (*.ads, *.adb) → gnatprove → Why3 Files (*.mlw) → Why3

Development of security-critical software
Spark Toolchain

Source Files (*.ads, *.adb) → gnatprove → Why3 Files (*.mlw) → Why3 → SMT Files

Isabelle-Why3 Files (*.xml)
Spark Toolchain

Source Files (*.ads, *.adb) → gnatprove → Why3 Files (*.mlw) → Why3 → Isabelle-Why3 Wrapper → Isabelle-Why3 Files (*.xml) → SMT Solver → SMT Files
SPARK Toolchain

Source Files (*.ads, *.adb) → gnatprove → Why3 Files (*.mlw) → Why3

Theory Files (*.thy) ← Isabelle-Why3 Wrapper ← Isabelle-Why3 Files (*.xml)

SMT Solver → SMT Files
Spark Toolchain

Source Files (*.ads, *.adb) → gnatprove → Why3 Files (*.mlw) → Why3

SMT Solver

SMT Files

Isabelle-Why3 Files (*.xml)

Isabelle-Why3 Wrapper

Theory Files (*.thy)

Development of security-critical software
Example: Euclidean Algorithm

```haskell
function GCD_Spec (M, N : Natural) return Natural
    with Ghost, Import;

function Euclid (M, N : Natural) return Natural
    with Post => Euclid’Result = GCD_Spec (M, N)
is
    A, B, R : Natural;
begin
    A := M; B := N;

    loop
        pragma Loop_Invariant (GCD_Spec (A, B) = GCD_Spec (M, N));
        exit when B = 0;
        R := A mod B;
        A := B; B := R;
    end loop;

    return A;
end Euclid;
```
Managing VCs using Why3 IDE
Proving VCs using Isabelle

```
theory gcd_Gcd__euclid__subprogram_def_WP_parameter_def_1
imports Why3
begin

why3_consts
  Gcd__gcd_spec.gcd_spec = gcd

why3_open "gcd_Gcd__euclid__subprogram_def_WP_parameter_def_1.xml"

why3_vc WP_parameter_def
  using `b1 ≠ 0` `in_range b1` `gcd a1 b1 = gcd m n`
  by (simp add: mod_def in_range_def Gcd_euclidean_mod)

proof (prove)
  goal (1 subgoal):
  1. gcd b1 (WP_parameter_def.mod a1 b1) = gcd m n
```
why3_open  Load and parse VCs
Commands of Why3-Plugin for Isabelle

why3_open  Load and parse VCs
why3_vc    Start proof of VC
Commands of Why3-Plugin for Isabelle

**why3_open**  Load and parse VCs

**why3_vc**   Start proof of VC

**why3_end**  Close Why3 environment
Commands of Why3-Plugin for Isabelle

- `why3_open` Load and parse VCs
- `why3_vc` Start proof of VC
- `why3_end` Close Why3 environment
- `why3_status` Show VCs
Commands of Why3-Plugin for Isabelle

- **why3_open**  Load and parse VCs
- **why3_vc**    Start proof of VC
- **why3_end**   Close Why3 environment
- **why3_status** Show VCs
- **why3consts** Link uninterpreted Why3 constants with Isabelle constants

(why3_consts etc. can be viewed as light-weight on-the-fly variant of Why3's realizations that happen completely on the Isabelle side)
Commands of Why3-Plugin for Isabelle

why3_open  Load and parse VCs
why3_vc    Start proof of VC
why3_end   Close Why3 environment
why3_status Show VCs
why3_consts Link uninterpreted Why3 constants with Isabelle constants
why3_types Link Why3 types with Isabelle types
Commands of Why3-Plugin for Isabelle

`why3_open` Load and parse VCs
`why3_vc` Start proof of VC
`why3_end` Close Why3 environment
`why3_status` Show VCs
`why3_consts` Link uninterpreted Why3 constants with Isabelle constants
`why3_types` Link Why3 types with Isabelle types
  (works for uninterpreted or data types)
Commands of Why3-Plugin for Isabelle

why3_open  Load and parse VCs
why3_vc    Start proof of VC
why3_end   Close Why3 environment
why3_status Show VCs
why3consts Link uninterpreted Why3 constants with Isabelle constants
why3_types Link Why3 types with Isabelle types
             (works for uninterpreted or data types)
why3 defs  Replace Why3 definitions by Isabelle definitions
Commands of Why3-Plugin for Isabelle

why3_open  Load and parse VCs
why3_vc    Start proof of VC
why3_end   Close Why3 environment
why3_status Show VCs
why3_consts Link uninterpreted Why3 constants with Isabelle constants
why3_types Link Why3 types with Isabelle types
  (works for uninterpreted or data types)
why3_defs  Replace Why3 definitions by Isabelle definitions
why3_thms  Replace Why3 axioms by Isabelle theorems
Commands of Why3-Plugin for Isabelle

`why3_open` Load and parse VCs

`why3_vc` Start proof of VC

`why3_end` Close Why3 environment

`why3_status` Show VCs

`why3_consts` Link uninterpreted Why3 constants with Isabelle constants

`why3_types` Link Why3 types with Isabelle types
  (works for uninterpreted or data types)

`why3_defs` Replace Why3 definitions by Isabelle definitions

`why3_thms` Replace Why3 axioms by Isabelle theorems

`why3_consts` etc. can be viewed as light-weight on-the-fly variant of Why3’s realizations that happen completely on the Isabelle side
Agenda

1. Introduction

2. A Link between Why3 and Isabelle

3. Applications

4. Conclusion
libsparkcrypto – A Cryptographic Library for SPARK

Supported algorithms
Supported algorithms

- Hash functions: (HMAC-)SHA-256/384/512
Supported algorithms

- Hash functions: (HMAC-)SHA-256/384/512
- Symmetric: AES-128/192/256
Supported algorithms

- Hash functions: (HMAC-)SHA-256/384/512
- Symmetric: AES-128/192/256
- Asymmetric: Elliptic Curves
libsparkcrypto – A Cryptographic Library for Spark

Supported algorithms

- Hash functions: (HMAC-)SHA-256/384/512
- Symmetric: AES-128/192/256
- Asymmetric: Elliptic Curves

Open Source

http://git.codelabs.ch/?p=spark-crypto.git
Layers of Cryptographic Functions

Big numbers

Modular multiplication, addition, subtraction
Layers of Cryptographic Functions

Elliptic curves (basic)  Point addition and doubling
Big numbers  Modular multiplication, addition, subtraction
Layers of Cryptographic Functions

- **Elliptic curves (derived)**: Scalar multiplication
- **Elliptic curves (basic)**: Point addition and doubling
- **Big numbers**: Modular multiplication, addition, subtraction
## Layers of Cryptographic Functions

<table>
<thead>
<tr>
<th>Security protocols</th>
<th>ECDSA, ECDH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elliptic curves (derived)</td>
<td>Scalar multiplication</td>
</tr>
<tr>
<td>Elliptic curves (basic)</td>
<td>Point addition and doubling</td>
</tr>
<tr>
<td>Big numbers</td>
<td>Modular multiplication, addition, subtraction</td>
</tr>
</tbody>
</table>
package Bignum
is

function Base return Math_Int.Math_Int is (Math_Int.From_Word32 (2) ** 32)
  with Ghost;

subtype Big_Int_Range is Natural range Natural’First .. Natural’Last - 1;

type Big_Int is array (Big_Int_Range range <>) of Types.Word32;

function Num_Of_Big_Int (A : Big_Int; F, L : Natural)
  return Math_Int.Math_Int
  with Ghost, Import, Global => null;

function Num_Of_Boolean (B : Boolean) return Math_Int.Math_Int
  with Ghost, Import, Global => null;

function Inverse (M, A : Math_Int.Math_Int) return Math_Int.Math_Int
  with Ghost, Import, Global => null;

end Bignum;
Formalization of Big Numbers in Isabelle

Abstraction function

\[ \text{num-of-big-int} : \mathbb{N} \to \mathbb{N} \]

\[ \text{num-of-big-int}(A, k, i) = (\exists j \in \{0, \ldots, i\}. \text{Base}_j \cdot A(k + j)) \]

Summation property

\[ \text{num-of-big-int}(A, k, i + j) = \text{num-of-big-int}(A, k, i) + \text{Base}_i \cdot \text{num-of-big-int}(A, k + i, j) \]

Modular inverse

\[ \text{minv} : \mathbb{N} \to \mathbb{N} \]

\[ \text{coprime}(x, m) = (0 < x < 1 < m) \land x \cdot \text{minv}(m, x) \mod m = 1 \]
Formalization of Big Numbers in Isabelle

Abstraction function

\[\text{num-of-big-int} :: (\text{int} \rightarrow \text{int}) \rightarrow \text{int} \rightarrow \text{int} \rightarrow \text{int}\]

\[\text{num-of-big-int} A k i = (\sum j = 0..<i. \ Base j \ast A (k + j))\]
Formalization of Big Numbers in Isabelle

Abstraction function

\[ \text{num-of-big-int :: (int \Rightarrow int) \Rightarrow int \Rightarrow int \Rightarrow int} \]
\[ \text{num-of-big-int A k i = } (\sum j = 0..<i. \text{Base}^j \ast A (k + j)) \]

Summation property

\[ \text{num-of-big-int A k (i + j) =} \]
\[ \text{num-of-big-int A k i + Base}^i \ast \text{num-of-big-int A (k + i) j} \]
Abstraction function

\[
\text{num-of-big-int} :: (\text{int} \Rightarrow \text{int}) \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{int}
\]
\[
\text{num-of-big-int} \ A \ k \ i = \left( \sum j = 0..<i. \ Base^j \ast A (k + j) \right)
\]

Summation property

\[
\text{num-of-big-int} \ A \ k \ (i + j) = \text{num-of-big-int} \ A \ k \ i + Base^i \ast \text{num-of-big-int} \ A \ (k + i) \ j
\]

Modular inverse

\[
\text{minv} :: \text{int} \Rightarrow \text{int} \Rightarrow \text{int}
\]
\[
\text{coprime} \ x \ m \implies 0 < x \implies 1 < m \implies x \ast \text{minv} \ m \ x \mod m = 1
\]
Montgomery Multiplication

Computes $x \cdot y = x \cdot y \cdot R \mod m$ where $R = b^n$.

Perform computations on numbers in Montgomery format

$e_x = x \cdot R \mod m$ (likewise for $e_y$).

Multiplication of numbers in Montgomery format

$e_x \cdot e_y = x \cdot R \cdot y \cdot R \cdot R_1 \mod m = x \cdot y \cdot R \mod m = e_x e_y \cdot 1 = x \cdot R \cdot 1 \cdot R_1 \mod m = x \mod m$.

Conversion between standard and Montgomery format

$x \cdot R_2 (R_2 \mod m) = x \cdot R_2 \cdot R_1 \mod m = e_x e_x \cdot 1 = x \cdot R \cdot 1 \cdot R_1 \mod m = x \mod m$. 
Montgomery Multiplication

Computes

\[ x \otimes y = x \cdot y \cdot R^{-1} \mod m \]

where \( R = b^n \)
Montgomery Multiplication

Computes

\[ x \otimes y = x \cdot y \cdot R^{-1} \mod m \]

where \( R = b^n \)

Perform computations on numbers in Montgomery format

\[ \tilde{x} = x \cdot R \mod m \]  (likewise for \( \tilde{y} \))
Montgomery Multiplication

Computes

\[ x \otimes y = x \cdot y \cdot R^{-1} \mod m \]

where \( R = b^n \)

Perform computations on numbers in Montgomery format

\[ \tilde{x} = x \cdot R \mod m \]

(likewise for \( \tilde{y} \))

Multiplication of numbers in Montgomery format

\[ \tilde{x} \otimes \tilde{y} = x \cdot R \cdot y \cdot R \cdot R^{-1} \mod m = x \cdot y \cdot R \mod m = \tilde{x} \cdot \tilde{y} \]
Montgomery Multiplication

Computes

\[ x \otimes y = x \cdot y \cdot R^{-1} \mod m \]

where \( R = b^n \)

Perform computations on numbers in Montgomery format

\[ \tilde{x} = x \cdot R \mod m \]  \( \text{(likewise for } \tilde{y}) \)

Multiplication of numbers in Montgomery format

\[ \tilde{x} \otimes \tilde{y} = x \cdot R \cdot y \cdot R \cdot R^{-1} \mod m = x \cdot y \cdot R \mod m = \tilde{x} \tilde{y} \]

Conversion between standard and Montgomery format
Montgomery Multiplication

Computes

\[ x \otimes y = x \cdot y \cdot R^{-1} \mod m \quad \text{where } R = b^n \]

Perform computations on numbers in Montgomery format

\[ \tilde{x} = x \cdot R \mod m \quad \text{(likewise for } \tilde{y}) \]

Multiplication of numbers in Montgomery format

\[ \tilde{x} \otimes \tilde{y} = x \cdot R \cdot y \cdot R \cdot R^{-1} \mod m = x \cdot y \cdot R \mod m = \tilde{x} \tilde{y} \]

Conversion between standard and Montgomery format

\[ x \otimes (R^2 \mod m) = x \cdot R^2 \cdot R^{-1} \mod m = x \cdot R \mod m = \tilde{x} \]
Montgomery Multiplication

Computes

\[ x \otimes y = x \cdot y \cdot R^{-1} \mod m \quad \text{where} \quad R = b^n \]

Perform computations on numbers in Montgomery format

\[ \tilde{x} = x \cdot R \mod m \quad \text{(likewise for } \tilde{y}) \]

Multiplication of numbers in Montgomery format

\[ \tilde{x} \otimes \tilde{y} = x \cdot R \cdot y \cdot R \cdot R^{-1} \mod m = x \cdot y \cdot R \mod m = \tilde{x} \tilde{y} \]

Conversion between standard and Montgomery format

\[ x \otimes (R^2 \mod m) = x \cdot R^2 \cdot R^{-1} \mod m = x \cdot R \mod m = \tilde{x} \]
\[ \tilde{x} \otimes 1 = x \cdot R \cdot 1 \cdot R^{-1} \mod m = x \mod m \]
Efficiently Computing $456 \cdot 789 \mod 987$

\[
a \leftarrow 0
\]

\[
\text{for } i = 0 \text{ to } n - 1 \text{ do}
\]

\[
u \leftarrow (a_0 + x_i \cdot y_0) \cdot \left(-m_0^{-1}\right) \mod b
\]

\[
a \leftarrow (a + x_i \cdot y + u \cdot m)/b
\]

\[
\text{end for}
\]

\[
\text{if } a \geq m \text{ then}
\]

\[
a \leftarrow a - m
\]

\[
\text{end if}
\]

\[
-7^{-1} \mod 10 = 7
\]
Efficiently Computing $456 \cdot 789 \mod 987$

\[
a \leftarrow 0
\]

\[
\text{for } i = 0 \text{ to } n - 1 \text{ do }
\]

\[
u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b
\]

\[
a \leftarrow (a + x_i \cdot y + u \cdot m)/b
\]

\[
\text{end for}
\]

\[
\text{if } a \geq m \text{ then}
\]

\[
a \leftarrow a - m
\]

\[
\text{end if}
\]

\[
-7^{-1} \mod 10 = 7
\]
Efficiently Computing $456 \cdot 789 \mod 987$

$$a \leftarrow 0$$

for $i = 0$ to $n - 1$ do

$$u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b$$

$$a \leftarrow (a + x_i \cdot y + u \cdot m) / b$$

end for

if $a \geq m$ then

$$a \leftarrow a - m$$

end if

$-7^{-1} \mod 10 = 7$
Efficiently Computing $456 \cdot 789 \mod 987$

\[
a \leftarrow 0
\]

for $i = 0$ to $n - 1$
  \[
u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b
\]
  \[
a \leftarrow (a + x_i \cdot y + u \cdot m)/b
\]

end for

if $a \geq m$
  \[
a \leftarrow a - m
\]

end if

$-7^{-1} \mod 10 = 7$
Efficiently Computing $456 \cdot 789 \mod 987$

\[ a \leftarrow 0 \]
\[ \text{for } i = 0 \text{ to } n - 1 \text{ do} \]
\[ u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b \]
\[ a \leftarrow (a + x_i \cdot y + u \cdot m)/b \]
\[ \text{end for} \]
\[ \text{if } a \geq m \text{ then} \]
\[ a \leftarrow a - m \]
\[ \text{end if} \]

\[ -7^{-1} \mod 10 = 7 \]
### Efficiently Computing $456 \cdot 789 \mod 987$

<table>
<thead>
<tr>
<th>$a$</th>
<th>$0$</th>
<th>$6 \cdot 789$</th>
<th>$4734$</th>
<th>$4734$ +</th>
<th>$8 \cdot 987$</th>
<th>$7896$ =</th>
<th>$12630$ / $10$ =</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \leftarrow 0$</td>
<td></td>
<td>$6 \cdot 789$</td>
<td>$4734$</td>
<td></td>
<td>$8 \cdot 987$</td>
<td>$7896$</td>
<td></td>
</tr>
<tr>
<td>$\text{for } i = 0 \text{ to } n - 1 \text{ do}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a \leftarrow (a + x_i \cdot y + u \cdot m) / b$</td>
<td></td>
<td>$6 \cdot 789$</td>
<td>$4734$</td>
<td></td>
<td>$8 \cdot 987$</td>
<td>$7896$</td>
<td></td>
</tr>
<tr>
<td>$\text{end for}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{if } a \geq m \text{ then}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a \leftarrow a - m$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{end if}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$-7^{-1} \mod 10 = 7$
Efficiently Computing \(456 \cdot 789 \mod 987\)

\[
\begin{align*}
    a & \leftarrow 0 \\
    \text{for } i = 0 \text{ to } n - 1 \text{ do} & \\
    & u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b \\
    & a \leftarrow (a + x_i \cdot y + u \cdot m)/b \\
    \text{end for} & \\
    \text{if } a \geq m \text{ then} & \\
    & a \leftarrow a - m \\
    \text{end if} & \\
\end{align*}
\]

\[-7^{-1} \mod 10 = 7\]
Efficiently Computing $456 \cdot 789 \mod 987$

\[
a \leftarrow 0\\
\text{for } i = 0 \text{ to } n - 1 \text{ do}\\
\quad u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b\\
\quad a \leftarrow (a + x_i \cdot y + u \cdot m) / b\\
\text{end for}\\
\text{if } a \geq m \text{ then}\\
\quad a \leftarrow a - m\\
\text{end if}\\
\]

$-7^{-1} \mod 10 = 7$
### Efficiently Computing $456 \cdot 789 \mod 987$

```
a \leftarrow 0
for \ i = 0 \ to \ n - 1 \ do
    u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b
    a \leftarrow (a + x_i \cdot y + u \cdot m)/b
end for
if \ a \geq m \ then
    a \leftarrow a - m
end if
```

\[
-7^{-1} \mod 10 = 7
\]
### Efficiently Computing $456 \cdot 789 \mod 987$

<table>
<thead>
<tr>
<th>$a \leftarrow 0$</th>
<th>$6 \cdot 789$</th>
<th>$4734$</th>
<th>$=$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\textbf{for } i = 0 \textbf{ to } n - 1 \textbf{ do}$</td>
<td>$8 \cdot 987$</td>
<td>$7896$</td>
<td>$=$</td>
</tr>
<tr>
<td>$u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b$</td>
<td>$12630$</td>
<td>$1263$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a \leftarrow (a + x_i \cdot y + u \cdot m) / b$</td>
<td>$5 \cdot 789$</td>
<td>$3945$</td>
<td>$=$</td>
</tr>
<tr>
<td>$\textbf{end for}$</td>
<td>$6 \cdot 987$</td>
<td>$5922$</td>
<td>$=$</td>
</tr>
<tr>
<td>$\textbf{if } a \geq m \textbf{ then}$</td>
<td>$=$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a \leftarrow a - m$</td>
<td>$=$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\textbf{end if}$</td>
<td>$=$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$-7^{-1} \mod 10 = 7$
Efficiently Computing $456 \cdot 789 \mod 987$

\[
a \leftarrow 0
\]

\[
\text{for } i = 0 \text{ to } n - 1 \text{ do}
\]

\[
u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b
\]

\[
a \leftarrow (a + x_i \cdot y + u \cdot m)/b
\]

\[
\text{end for}
\]

\[
\text{if } a \geq m \text{ then}
\]

\[
a \leftarrow a - m
\]

\[
\text{end if}
\]

\[
-7^{-1} \mod 10 = 7
\]
Efficiently Computing $456 \cdot 789 \mod 987$

\[
a \leftarrow 0
\]

\[
\text{for } i = 0 \text{ to } n - 1 \text{ do}
\]

\[
u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b
\]

\[
a \leftarrow (a + x_i \cdot y + u \cdot m)/b
\]

\[
\text{end for}
\]

\[
\text{if } a \geq m \text{ then}
\]

\[
a \leftarrow a - m
\]

\[
\text{end if}
\]

\[
-7^{-1} \mod 10 = 7
\]
Efficiently Computing $456 \cdot 789 \mod 987$

$$a \leftarrow 0$$

```
for i = 0 to n - 1 do
    u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b
    a \leftarrow (a + x_i \cdot y + u \cdot m)/b
end for
if a \geq m then
    a \leftarrow a - m
end if
```

$$-7^{-1} \mod 10 = 7$$

<table>
<thead>
<tr>
<th>$a$</th>
<th>$6 \cdot 789$</th>
<th>$4734$</th>
<th>$4734$</th>
<th>$4734$</th>
<th>$4734$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$8 \cdot 987$</td>
<td>$7896$</td>
<td>$12630$</td>
<td>$12630$</td>
<td>$12630$</td>
</tr>
<tr>
<td></td>
<td>$5 \cdot 789$</td>
<td>$3945$</td>
<td>$5208$</td>
<td>$5208$</td>
<td>$5208$</td>
</tr>
<tr>
<td></td>
<td>$6 \cdot 987$</td>
<td>$5922$</td>
<td>$11130$</td>
<td>$11130$</td>
<td>$11130$</td>
</tr>
<tr>
<td></td>
<td>$4 \cdot 789$</td>
<td>$3156$</td>
<td>$3156$</td>
<td>$3156$</td>
<td>$3156$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Efficiently Computing $456 \cdot 789 \mod 987$

\begin{align*}
a & \leftarrow 0 \\
\text{for } i = 0 \text{ to } n - 1 \text{ do} & \\
\quad u & \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b \\
\quad a & \leftarrow (a + x_i \cdot y + u \cdot m) / b \\
\text{end for} & \\
\text{if } a \geq m \text{ then} & \\
\quad a & \leftarrow a - m \\
\text{end if} & \\
-7^{-1} \mod 10 &= 7
\end{align*}

\begin{tabular}{|c|c|c|c|}
\hline
& 0 & + & \\
\hline
$6 \cdot 789$ & 4734 & = & \\
\hline
$8 \cdot 987$ & 7896 & = & \\
\hline
$12630 / 10$ & 1263 & + & \\
\hline
$5 \cdot 789$ & 3945 & = & \\
\hline
$5208$ & + & \\
\hline
$6 \cdot 987$ & 5922 & = & \\
\hline
$11130 / 10$ & 1113 & + & \\
\hline
$4 \cdot 789$ & 3156 & = & \\
\hline
$4269$ & + & \\
\hline
\end{tabular}
Efficiently Computing $456 \cdot 789 \mod 987$

$a \leftarrow 0$

for $i = 0 \text{ to } n-1$ do

\[
u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b
\]

\[
a \leftarrow (a + x_i \cdot y + u \cdot m)/b
\]

end for

if $a \geq m$ then

\[
a \leftarrow a - m
\]

end if

\[-7^{-1} \mod 10 = 7\]
Efficiently Computing $456 \cdot 789 \mod 987$

\[
\begin{align*}
a & \leftarrow 0 \\
\text{for } i = 0 \text{ to } n-1 \text{ do} & \\
\quad u & \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b \\
\quad a & \leftarrow (a + x_i \cdot y + u \cdot m) / b \\
\text{end for} \\
\text{if } a \geq m \text{ then} & \\
\quad a & \leftarrow a - m \\
\text{end if} \\
-7^{-1} \mod 10 = 7
\end{align*}
\]
Efficiently Computing $456 \cdot 789 \mod 987$

\[
a \leftarrow 0
\]
\[
\text{for } i = 0 \text{ to } n - 1 \text{ do}
\]
\[
u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b
\]
\[
a \leftarrow (a + x_i \cdot y + u \cdot m)/b
\]
\[
\text{end for}
\]
\[
\text{if } a \geq m \text{ then}
\]
\[
a \leftarrow a - m
\]
\[
\text{end if}
\]

\[-7^{-1} \mod 10 = 7\]
Efficiently Computing $456 \cdot 789 \mod 987$

\[
a \leftarrow 0
\]

\[
\text{for } i = 0 \text{ to } n - 1 \text{ do}
\]

\[
u \leftarrow (a_0 + x_i \cdot y_0) \cdot -m_0^{-1} \mod b
\]

\[
a \leftarrow (a + x_i \cdot y + u \cdot m) / b
\]

\[
\text{end for}
\]

\[
\text{if } a \geq m \text{ then}
\]

\[
a \leftarrow a - m
\]

\[
\text{end if}
\]

\[-7^{-1} \mod 10 = 7\]

\[
723 \cdot 1000 \mod 987 = 516 = 456 \cdot 789 \mod 987
\]
Montgomery Multiplication in Spark

for I in Natural range A_First .. A_Last loop
  Carry1 := 0; Carry2 := 0;
  XI := X (X_First + (I - A_First));
  U := (A (A_First) + XI * Y (Y_First)) * M_Inv;
  Single_Add_Mult_Mult
    (A (A_First), XI, Y (Y_First),
     M (M_First), U, Carry1, Carry2);
  Add_Mult_Mult
    (A, A_First, A_Last - 1,
     Y, Y_First + 1, M, M_First + 1,
     XI, U, Carry1, Carry2);
  A (A_Last) := A_MSW + Carry1;
  A_MSW := Carry2 + Word_Of_Boolean (A (A_Last) < Carry1);
end loop;
Specification of Montgomery Multiplication

**Preconditions**

\[
\text{Num\_Of\_Big\_Int \ (Y, \ Y\_First, \ A\_Last - A\_First + 1) < Num\_Of\_Big\_Int \ (M, \ M\_First, \ A\_Last - A\_First + 1) and} \\
1 < \text{Num\_Of\_Big\_Int \ (M, \ M\_First, \ A\_Last - A\_First + 1) and} \\
1 + M\_Inv \times M \ (M\_First) = 0
\]
Specification of Montgomery Multiplication

**Preconditions**

\[
\text{Num\_Of\_Big\_Int} (Y, Y\_First, A\_Last - A\_First + 1) < \\
\text{Num\_Of\_Big\_Int} (M, M\_First, A\_Last - A\_First + 1) \text{ and} \\
1 < \text{Num\_Of\_Big\_Int} (M, M\_First, A\_Last - A\_First + 1) \text{ and} \\
1 + M\_Inv \times M (M\_First) = 0
\]

**Postcondition**

\[
\text{Num\_Of\_Big\_Int} (A, A\_First, A\_Last - A\_First + 1) = \\
(\text{Num\_Of\_Big\_Int} (X, X\_First, A\_Last - A\_First + 1) \times \\
\text{Num\_Of\_Big\_Int} (Y, Y\_First, A\_Last - A\_First + 1) \times \\
\text{Inverse} (\text{Num\_Of\_Big\_Int} (M, M\_First, A\_Last - A\_First + 1), \text{ Base}) \times (A\_Last - A\_First + 1)) \mod \\
\text{Num\_Of\_Big\_Int} (M, M\_First, A\_Last - A\_First + 1)
\]
Specification of Montgomery Multiplication

**Preconditions**

\[
\text{Num}_{\text{Of Big Int}} (Y, Y_{\text{First}}, A_{\text{Last}} - A_{\text{First}} + 1) < \\
\text{Num}_{\text{Of Big Int}} (M, M_{\text{First}}, A_{\text{Last}} - A_{\text{First}} + 1) \quad \text{and} \\
1 < \text{Num}_{\text{Of Big Int}} (M, M_{\text{First}}, A_{\text{Last}} - A_{\text{First}} + 1) \quad \text{and} \\
1 + M_{\text{Inv}} \times M (M_{\text{First}}) = 0
\]

**Postcondition**

\[
\text{Num}_{\text{Of Big Int}} (A, A_{\text{First}}, A_{\text{Last}} - A_{\text{First}} + 1) = \\
(\text{Num}_{\text{Of Big Int}} (X, X_{\text{First}}, A_{\text{Last}} - A_{\text{First}} + 1) \times \\
\text{Num}_{\text{Of Big Int}} (Y, Y_{\text{First}}, A_{\text{Last}} - A_{\text{First}} + 1) \times \\
\text{Inverse} \left( \text{Num}_{\text{Of Big Int}} (M, M_{\text{First}}, A_{\text{Last}} - A_{\text{First}} + 1), \\
\text{Base} \right) ** (A_{\text{Last}} - A_{\text{First}} + 1) \right) \mod \\
\text{Num}_{\text{Of Big Int}} (M, M_{\text{First}}, A_{\text{Last}} - A_{\text{First}} + 1)
\]

**In mathematical notation...**

\[a = x \cdot y \cdot b^{-n} \mod m\]
Elliptic Curves

Applications: ECDH (key agreement) and ECDSA (authentication)

Curves in Weierstrass form over fields of characteristic $> 2$

Primitive operation: point addition

Points on curve form a group wrt. point addition

Coordinate systems:
- Affine $(x, y)$
- Projective $(x/z, y/z)$

Point addition does not require computation of inverse

Abstract properties formalized in Isabelle based on Coq formalization by Laurent Théry

Proved correspondence of Spark implementation with abstract specification
Elliptic Curves

- Applications: ECDH (key agreement) and ECDSA (authentication)
Elliptic Curves

- Applications: ECDH (key agreement) and ECDSA (authentication)
- Curves in Weierstrass form over fields of characteristic $> 2$
Elliptic Curves

- Applications: ECDH (key agreement) and ECDSA (authentication)
- Curves in Weierstrass form over fields of characteristic $> 2$
- Primitive operation: point addition
Elliptic Curves

- Applications: ECDH (key agreement) and ECDSA (authentication)
- Curves in Weierstrass form over fields of characteristic $> 2$
- Primitive operation: point addition
- Points on curve form a group wrt. point addition
Elliptic Curves

- Applications: ECDH (key agreement) and ECDSA (authentication)
- Curves in Weierstrass form over fields of characteristic \( \geq 2 \)
- Primitive operation: point addition
- Points on curve form a group wrt. point addition
- Coordinate systems:
Elliptic Curves

- Applications: ECDH (key agreement) and ECDSA (authentication)
- Curves in Weierstrass form over fields of characteristic > 2
- Primitive operation: point addition
- Points on curve form a group wrt. point addition
- Coordinate systems:
  - Affine \((x, y)\)
Elliptic Curves

- Applications: ECDH (key agreement) and ECDSA (authentication)
- Curves in Weierstrass form over fields of characteristic $> 2$
- Primitive operation: point addition
- Points on curve form a group wrt. point addition
- Coordinate systems:
  - Affine $(x, y)$
  - Projective $(x, y, z)$
Elliptic Curves

- Applications: ECDH (key agreement) and ECDSA (authentication)
- Curves in Weierstrass form over fields of characteristic $> 2$
- Primitive operation: point addition
- Points on curve form a group wrt. point addition
- Coordinate systems:
  - Affine $(x, y)$
  - Projective $(x, y, z) \mapsto (x/z, y/z)$
Elliptic Curves

- Applications: ECDH (key agreement) and ECDSA (authentication)
- Curves in Weierstrass form over fields of characteristic $> 2$
- Primitive operation: point addition
- Points on curve form a group wrt. point addition
- Coordinate systems:
  - Affine $(x, y)$
  - Projective $(x, y, z) \mapsto (x/z, y/z)$
    point addition does not require computation of inverse
Elliptic Curves

- Applications: ECDH (key agreement) and ECDSA (authentication)
- Curves in Weierstrass form over fields of characteristic $> 2$
- Primitive operation: point addition
- Points on curve form a group wrt. point addition
- Coordinate systems:
  - Affine $(x, y)$
  - Projective $(x, y, z) \mapsto (x/z, y/z)$
    - point addition does not require computation of inverse
- Abstract properties formalized in Isabelle
Elliptic Curves

- Applications: ECDH (key agreement) and ECDSA (authentication)
- Curves in Weierstrass form over fields of characteristic $> 2$
- Primitive operation: point addition
- Points on curve form a group wrt. point addition
- Coordinate systems:
  - Affine $(x, y)$
  - Projective $(x, y, z) \mapsto (x/z, y/z)$
    point addition does not require computation of inverse
- Abstract properties formalized in Isabelle
  based on Coq formalization by Laurent Théry
Elliptic Curves

- Applications: ECDH (key agreement) and ECDSA (authentication)
- Curves in Weierstrass form over fields of characteristic $> 2$
- Primitive operation: point addition
- Points on curve form a group wrt. point addition
- Coordinate systems:
  - Affine $(x, y)$
  - Projective $(x, y, z) \rightarrow (x/z, y/z)$
    - point addition does not require computation of inverse
- Abstract properties formalized in Isabelle based on Coq formalization by Laurent Théry
- Proved correspondence of SPARK implementation with abstract specification

Page 26
Elliptic Curves – Abstract Specification

datatype 'a point = Infinity | Point 'a 'a

locale ell-field = field +
  assumes two-not-zero: 2 ≠ 0

definition nonsingular :: 'a ⇒ 'a ⇒ bool where
  nonsingular a b = (4 ⊗ a ↑ 3 ⊕ 27 ⊗ b ↑ 2 ≠ 0)

definition on-curve :: 'a ⇒ 'a ⇒ 'a point ⇒ bool where
  on-curve a b p = (case p of
    Infinity ⇒ True
    | Point x y ⇒ x ∈ carrier R ∧ y ∈ carrier R ∧
      y ↑ 2 = x ↑ 3 ⊕ a ⊗ x ⊕ b)
Point Addition

**definition** add :: 'a ⇒ 'a point ⇒ 'a point ⇒ 'a point where

\[
\text{add } a \ p_1 \ p_2 = \begin{cases} 
\text{Infinity} & \Rightarrow \ p_2 \\
\text{Point } x_1 \ y_1 & \Rightarrow \begin{cases} 
\text{Infinity} & \Rightarrow \ p_1 \\
\text{Point } x_2 \ y_2 & \Rightarrow \begin{cases} 
\text{if } x_1 = x_2 \text{ then} & \\
\text{if } y_1 = \ominus y_2 \text{ then } \text{Infinity} & \\
\text{else let } l = (\langle 3 \rangle \otimes x_1 \uparrow 2 \ominus a) \odot (\langle 2 \rangle \otimes y_1); \\
\quad x_3 = l \uparrow 2 \ominus \langle 2 \rangle \otimes x_1 \\
\quad \text{in Point } x_3 (\ominus y_1 \ominus l \otimes (x_3 \ominus x_1)) & \\
\text{else let } l = (y_2 \ominus y_1) \odot (x_2 \ominus x_1); \\
\quad x_3 = l \uparrow 2 \ominus x_1 \ominus x_2 \\
\quad \text{in Point } x_3 (\ominus y_1 \ominus l \otimes (x_3 \ominus x_1))) & \end{cases} \\
\end{cases}
\end{cases}
\]

---

Page 28 | 30.5.2017 | Development of security-critical software
Point Addition – Properties

**Lemma** \(\text{add-closed}:\)**

**Assumes** \(a \in \text{carrier } R \land b \in \text{carrier } R\)**

**And** \(\text{on-curve } a \ b \ p_1 \ \land \ \text{on-curve } a \ b \ p_2\)**

**Shows** \(\text{on-curve } a \ b \ (\text{add } a \ p_1 \ p_2)\)**
**Point Addition – Properties**

**lemma** *add-closed:*

*assumes* \( a \in \text{carrier } R \text{ and } b \in \text{carrier } R \)  
*and* on-curve \( a \ b \ p_1 \text{ and on-curve } a \ b \ p_2 \)  
*shows* on-curve \( a \ b \ (add \ a \ p_1 \ p_2) \)

**lemma** *add-comm:*

*assumes* \( a \in \text{carrier } R \text{ and } b \in \text{carrier } R \)  
*and* on-curve \( a \ b \ p_1 \text{ and on-curve } a \ b \ p_2 \)  
*shows* \( add \ a \ p_1 \ p_2 = add \ a \ p_2 \ p_1 \)
Point Addition – Properties

**lemma** add-closed:
- assumes \( a \in \text{carrier } R \) and \( b \in \text{carrier } R \)
- and on-curve \( a \ b \ p_1 \) and on-curve \( a \ b \ p_2 \)
- shows on-curve \( a \ b \) \((\text{add } a \ p_1 \ p_2)\)

**lemma** add-comm:
- assumes \( a \in \text{carrier } R \) and \( b \in \text{carrier } R \)
- and on-curve \( a \ b \ p_1 \) and on-curve \( a \ b \ p_2 \)
- shows \( \text{add } a \ p_1 \ p_2 = \text{add } a \ p_2 \ p_1 \)

**lemma** add-assoc:
- assumes \( a: a \in \text{carrier } R \) and \( b: b \in \text{carrier } R \)
- and \( ab: \) nonsingular \( a \ b \)
- and \( p_1: \) on-curve \( a \ b \ p_1 \) and \( p_2: \) on-curve \( a \ b \ p_2 \)
- and \( p_3: \) on-curve \( a \ b \ p_3 \)
- shows \( \text{add } a \ p_1 \ (\text{add } a \ p_2 \ p_3) = \text{add } a \ (\text{add } a \ p_1 \ p_2) \ p_3 \)
type-synonym 'a ppoint = 'a × 'a × 'a

definition (in field) make-affine :: 'a ppoint ⇒ 'a point where
make-affine p =
  (let (x, y, z) = p
   in if z = 0 then Infinity else Point (x ⊙ z) (y ⊙ z))

lemma (in ell-field) padd-correct:
  assumes a: a ∈ carrier R and b: b ∈ carrier R
  and p₁: on-curvep a b p₁ and p₂: on-curvep a b p₂
  shows make-affine (padd a p₁ p₂) =
    add a (make-affine p₁) (make-affine p₂)
**Projective Coordinates – Equality**

**definition (in cring) proj-eq :: 'a ppoint ⇒ 'a ppoint ⇒ bool**

where

\[
\text{proj-eq} = (\lambda(x_1, y_1, z_1) (x_2, y_2, z_2). \\
(z_1 = 0) = (z_2 = 0) \land \\
x_1 \otimes z_2 = x_2 \otimes z_1 \land y_1 \otimes z_2 = y_2 \otimes z_1)
\]

**lemma (in field) make-affine-proj-eq-iff:**

\[\text{in-carrierp } p \implies \text{in-carrierp } p' \implies \]

\[\text{proj-eq } p \; p' = (\text{make-affine } p = \text{make-affine } p')\]
Function PointAddSpec

```plaintext
function Point_Add_Spec
    (M, A, X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3 : Math_Int.Math_Int)
    return Boolean
    with Ghost, Import, Global => null;

procedure Point_Add
    (X1, Y1, Z1 : in Bignum.Big_Int;
     X2, Y2, Z2 : in Bignum.Big_Int;
     X3, Y3, Z3 : out Bignum.Big_Int;
     ...
     A : in Bignum.Big_Int;
     M : in Bignum.Big_Int;
     M_Inv : in Types.Word32)
    with Depends => ...
    Pre => ...
    Post =>
        Point_Add_Spec
            (Bignum.Num_Of_Big_Int (M, M_First, X1_Last - X1_First + 1), ...
```
**Specification of Point Addition – Isabelle Part**

**definition** \texttt{point-add-spec :: \textit{math-int} \Rightarrow \textit{math-int} \Rightarrow \textit{math-int} \Rightarrow \textit{math-int} \Rightarrow \textit{math-int} \Rightarrow \textit{math-int} \Rightarrow \textit{math-int} \Rightarrow \textit{math-int} \Rightarrow \textit{bool}}

**where**

\texttt{point-add-spec \textit{m} \textit{a} \textit{x} \textit{y} \textit{z} \textit{1} \textit{x} \textit{2} \textit{y} \textit{2} \textit{z} \textit{2} \textit{x} \textit{3} \textit{y} \textit{3} \textit{z} \textit{3} =}

\begin{align*}
(\textit{let } \textit{r} &= \textit{residue-ring} (\textit{int-of-math-int} \textit{m}); \\
\textit{a'} &= \textit{int-of-math-int} \textit{a} \mod \textit{int-of-math-int} \textit{m} \\
\text{in cring.proj-eq} \textit{r} \\
(\text{cring.padd} \textit{r} \textit{a'}) \\
(\textit{int-of-math-int} \textit{x} \textit{1}, \textit{int-of-math-int} \textit{y} \textit{1}, \textit{int-of-math-int} \textit{z} \textit{1}) \\
(\textit{int-of-math-int} \textit{x} \textit{2}, \textit{int-of-math-int} \textit{y} \textit{2}, \textit{int-of-math-int} \textit{z} \textit{2}) \\
(\textit{int-of-math-int} \textit{x} \textit{3}, \textit{int-of-math-int} \textit{y} \textit{3}, \textit{int-of-math-int} \textit{z} \textit{3}))
\end{align*}

**why3 consts**

\texttt{Lsc--ec--point-add-spec.point-add-spec = point-add-spec}
Agenda

1. Introduction

2. A Link between Why3 and Isabelle

3. Applications

4. Conclusion
Achievements

- Correctness of complex mathematical algorithms can be proved using SPARK
Achievements

- Correctness of complex mathematical algorithms can be proved using Spark
- Link with interactive prover allows to prove advanced properties that are beyond reach of automatic provers
Achievements

- Correctness of complex mathematical algorithms can be proved using Spark
- Link with interactive prover allows to prove advanced properties that are beyond reach of automatic provers
- Behaviour of programs can be specified in an abstract way
Achievements

- Correctness of complex mathematical algorithms can be proved using **SPARK**
- Link with interactive prover allows to prove advanced properties that are beyond reach of automatic provers
- Behaviour of programs can be specified in an abstract way
- **SPARK** implementation can be shown to correspond to abstract specification
Challenges

- Why3 model not really suitable for human consumption
Challenges

- Why3 model not really suitable for human consumption
- Translation to Why3 introduces many axioms (must be realized in Isabelle)
Challenges

- Why3 model not really suitable for human consumption
- Translation to Why3 introduces many axioms (must be realized in Isabelle)
- Why3 session file not perfectly suitable for interactive provers
Challenges

- Why3 model not really suitable for human consumption
- Translation to Why3 introduces many axioms (must be realized in Isabelle)
- Why3 session file not perfectly suitable for interactive provers e.g. goal matching algorithm sometimes gets confused by complex control flow
Ongoing and Future Work

- Verification of Muen separation kernel
Ongoing and Future Work

- Verification of Muen separation kernel
  - Runtime behaviour
Ongoing and Future Work

- Verification of Muen separation kernel
  - Runtime behaviour
    e.g. correct saving / restoring of subject states
Ongoing and Future Work

- Verification of Muen separation kernel
  - Runtime behaviour
    - e.g. correct saving / restoring of subject states
  - Memory layout

Improvement of Why3 plugin for Isabelle
- Automatic realization of axioms generated for Spark types

Continued verification of security-critical components and protocols
Ongoing and Future Work

- Verification of Muen separation kernel
  - Runtime behaviour
    - e.g. correct saving / restoring of subject states
  - Memory layout
    - e.g. correct construction of page tables
Ongoing and Future Work

- Verification of Muen separation kernel
  - Runtime behaviour
    e.g. correct saving / restoring of subject states
  - Memory layout
    e.g. correct construction of page tables
  - Non-interference properties
Ongoing and Future Work

- Verification of Muen separation kernel
  - Runtime behaviour
    e.g. correct saving / restoring of subject states
  - Memory layout
    e.g. correct construction of page tables
  - Non-interference properties
  - Abstract Isabelle model

Improvement of Why3 plugin for Isabelle
Automatic realization of axioms generated for Spark types
Continued verification of security-critical components and protocols
Ongoing and Future Work

- Verification of Muen separation kernel
  - Runtime behaviour
    - e.g. correct saving / restoring of subject states
  - Memory layout
    - e.g. correct construction of page tables
  - Non-interference properties
  - Abstract Isabelle model

- Improvement of Why3 plugin for Isabelle
Ongoing and Future Work

- Verification of Muen separation kernel
  - Runtime behaviour
    e.g. correct saving / restoring of subject states
  - Memory layout
    e.g. correct construction of page tables
  - Non-interference properties
  - Abstract Isabelle model

- Improvement of Why3 plugin for Isabelle
  Automatic realization of axioms generated for Spark types
Ongoing and Future Work

- Verification of Muen separation kernel
  - Runtime behaviour
    - e.g. correct saving / restoring of subject states
  - Memory layout
    - e.g. correct construction of page tables
  - Non-interference properties
  - Abstract Isabelle model

- Improvement of Why3 plugin for Isabelle
  - Automatic realization of axioms generated for Spark types

- Continued verification of security-critical components and protocols
secunet Security Networks AG

Stefan Berghofer

Kurfürstenstraße 58
45138 Essen
Tel.: +49-201-5454-3606
Fax: +49-201-5454-1323
stefan.berghofer@secunet.com
www.secunet.com