Why formal methods remains inaccessible for most cryptographers?

Georgio Nicolas (COSIC, KU Leuven)

Wednesday 27th March, 2024

EuroProofNet WG3 Meeting, TU Dresden

Table of contents

About Me

About Me 1/2

- 2nd year PhD Student, supervised by Nigel Smart and Bart Preneel.

About Me 1/2

- 2nd year PhD Student, supervised by Nigel Smart and Bart Preneel.
- Been dabbling with formal methods since 2018.

About Me 1/2

- 2nd year PhD Student, supervised by Nigel Smart and Bart Preneel.
- Been dabbling with formal methods since 2018.
- Worked on cute tools that are relatively usable: Noise Explorer / Verifpal.

About Me 2/2

- I give 2 lectures/year about formal methods in a cryptographic protocols course.

About Me 2/2

- I give 2 lectures/year about formal methods in a cryptographic protocols course.
- Had a masters student who worked on verifying parts of a Shamir Secret Sharing implementation in Rust.

About Me 2/2

- I give 2 lectures/year about formal methods in a cryptographic protocols course.
- Had a masters student who worked on verifying parts of a Shamir Secret Sharing implementation in Rust.
- My love for the computational model is unreciprocated by the tools.

About Me 2/2

- I give 2 lectures/year about formal methods in a cryptographic protocols course.
- Had a masters student who worked on verifying parts of a Shamir Secret Sharing implementation in Rust.
- My love for the computational model is unreciprocated by the tools.
- For some reason, everyone in my group thinks that I'm an expert at formal methods...

About Me 2/2

- I give 2 lectures/year about formal methods in a cryptographic protocols course.
- Had a masters student who worked on verifying parts of a Shamir Secret Sharing implementation in Rust.
- My love for the computational model is unreciprocated by the tools.
- For some reason, everyone in my group thinks that I'm an expert at formal methods...
- I'm definitely not.

Why am I giving this talk?

Cryptography (1/3)

- Cryptography is equally powerful and fragile.

Cryptography (1/3)

- Cryptography is equally powerful and fragile.
- High-assurances are required by definition.

Cryptography (1/3)

- Cryptography is equally powerful and fragile.
- High-assurances are required by definition.
- Constructions and proofs can get very complex.

Cryptography (1/3)

- Cryptography is equally powerful and fragile.
- High-assurances are required by definition.
- Constructions and proofs can get very complex.
- Perfect material for leveraging Formal Methods!

Disclaimer (2/3)

- Whatever I say here is not to discredit any of the amazing work done by the community!

Disclaimer (2/3)

- Whatever I say here is not to discredit any of the amazing work done by the community!
- Every single contribution was crucial to getting us where we are today.

Disclaimer (2/3)

- Whatever I say here is not to discredit any of the amazing work done by the community!
- Every single contribution was crucial to getting us where we are today.
- We have a plethora of tools with all sorts of functionalities and guarantees!

Disclaimer (2/3)

- Whatever I say here is not to discredit any of the amazing work done by the community!
- Every single contribution was crucial to getting us where we are today.
- We have a plethora of tools with all sorts of functionalities and guarantees!
- However... I think we can do better in some things.

Concerns (3/3)

- Resolving dependencies (for humans and software)
- Teaching material and documentation
- Reproducing and reasoning about results
- General friction with getting results accepted by the community

Concerns

Resolving Dependencies (for humans and software) (1/4)

- Our tools have dependency chains.

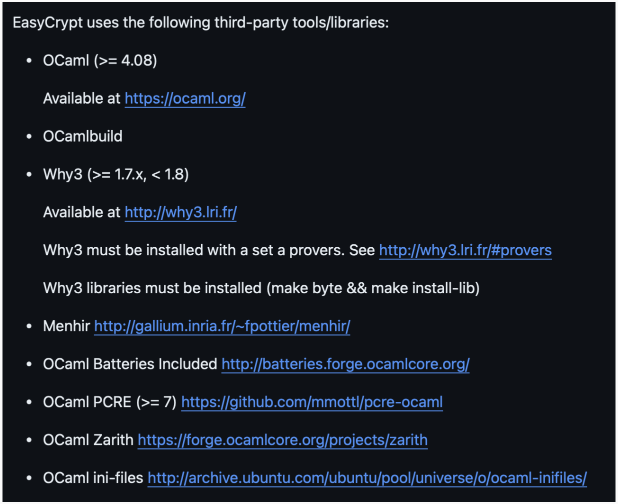
Resolving Dependencies (for humans and software) (1/4)

- Our tools have dependency chains.
- Managing software dependency chains can be painful.

Resolving Dependencies (for humans and software) (1/4)

- Our tools have dependency chains.
- Managing software dependency chains can be painful.
- Managing human dependency chains is even more painful.

Resolving Dependencies (for humans and software) (2/4)



EasyCrypt uses the following third-party tools/libraries:

- OCaml (>= 4.08)

  Available at https://ocaml.org/

- OCamlbuild
- Why3 (>= 1.7.x, < 1.8)

  Available at http://why3.lri.fr/

  Why3 must be installed with a set a provers. See http://why3.lri.fr/#provers

  Why3 libraries must be installed (make byte && make install-lib)

- Menhir http://gallium.inria.fr/~fpottier/menhir/
- OCaml Batteries Included http://batteries.forge.ocamlcore.org/
- OCaml PCRE (>= 7) https://github.com/mmottl/pcre-ocaml
- OCaml Zarith https://forge.ocamlcore.org/projects/zarith
- OCaml ini-files http://archive.ubuntu.com/ubuntu/pool/universe/o/ocaml-inifiles/

Figure 1: EasyCrypt Software Dependencies.

Resolving Dependencies (for humans and software) (3/4)

EasyCrypt Human Dependencies:

- Figuring out what needs to be in this list

Resolving Dependencies (for humans and software) (3/4)

EasyCrypt Human Dependencies:

- Figuring out what needs to be in this list
- Cryptography (and formal security definitions)

Resolving Dependencies (for humans and software) (3/4)

EasyCrypt Human Dependencies:

- Figuring out what needs to be in this list
- Cryptography (and formal security definitions)
- Game-based proofs

Resolving Dependencies (for humans and software) (3/4)

EasyCrypt Human Dependencies:

- Figuring out what needs to be in this list
- Cryptography (and formal security definitions)
- Game-based proofs
- Distributions and Probabilistic Logic

Resolving Dependencies (for humans and software) (3/4)

EasyCrypt Human Dependencies:

- Figuring out what needs to be in this list
- Cryptography (and formal security definitions)
- Game-based proofs
- Distributions and Probabilistic Logic
- Type Theory and Functional Programming

Resolving Dependencies (for humans and software) (3/4)

EasyCrypt Human Dependencies:

- Figuring out what needs to be in this list
- Cryptography (and formal security definitions)
- Game-based proofs
- Distributions and Probabilistic Logic
- Type Theory and Functional Programming
- Algebraic Structures

Resolving Dependencies (for humans and software) (3/4)

EasyCrypt Human Dependencies:

- Figuring out what needs to be in this list
- Cryptography (and formal security definitions)
- Game-based proofs
- Distributions and Probabilistic Logic
- Type Theory and Functional Programming
- Algebraic Structures
- Inductive/Deductive Reasoning

Resolving Dependencies (for humans and software) (3/4)

EasyCrypt Human Dependencies:

- Figuring out what needs to be in this list
- Cryptography (and formal security definitions)
- Game-based proofs
- Distributions and Probabilistic Logic
- Type Theory and Functional Programming
- Algebraic Structures
- Inductive/Deductive Reasoning
- Debugging

Resolving Dependencies (for humans and software) (3/4)

EasyCrypt Human Dependencies:

- Figuring out what needs to be in this list
- Cryptography (and formal security definitions)
- Game-based proofs
- Distributions and Probabilistic Logic
- Type Theory and Functional Programming
- Algebraic Structures
- Inductive/Deductive Reasoning
- Debugging
- Management of feelings when opam decides to not work

Resolving Dependencies (for humans and software) (3/4)

EasyCrypt Human Dependencies:

- Figuring out what needs to be in this list
- Cryptography (and formal security definitions)
- Game-based proofs
- Distributions and Probabilistic Logic
- Type Theory and Functional Programming
- Algebraic Structures
- Inductive/Deductive Reasoning
- Debugging
- Management of feelings when opam decides to not work
- Emacs...

Resolving Dependency Chains (for humans and software) (4/4)

Let's assume that such a list exists:

- Can we be sure that it is comprehensive?

Resolving Dependency Chains (for humans and software) (4/4)

Let's assume that such a list exists:

- Can we be sure that it is comprehensive?
- What if we learn all of this, and then realize that the research question depends on solving another problem?

Resolving Dependency Chains (for humans and software) (4/4)

Let's assume that such a list exists:

- Can we be sure that it is comprehensive?
- What if we learn all of this, and then realize that the research question depends on solving another problem?
- Can we plan for such a project if there is no direct access to an expert?

Teaching Material and Documentation

- Teaching material for generic proof assistant use is abundant!

Teaching Material and Documentation

- Teaching material for generic proof assistant use is abundant!
- Unfortunately, this is not the case for crypto-specific tools.

Teaching Material and Documentation

- Teaching material for generic proof assistant use is abundant!
- Unfortunately, this is not the case for crypto-specific tools.
- I understand that writing documentation does not produce papers for the person writing it;

Teaching Material and Documentation

- Teaching material for generic proof assistant use is abundant!
- Unfortunately, this is not the case for crypto-specific tools.
- I understand that writing documentation does not produce papers for the person writing it;
- but good documentation might enable more people to get into producing their own results!

Teaching Material and Documentation

- Teaching material for generic proof assistant use is abundant!
- Unfortunately, this is not the case for crypto-specific tools.
- I understand that writing documentation does not produce papers for the person writing it;
- but good documentation might enable more people to get into producing their own results!
- How can we do better at that? Where can we start?

Reproducing and reasoning about results

- When the constructions we are attempting to analyze are so complex, the results would be only understandable by experts using the same tool.

Reproducing and reasoning about results

- When the constructions we are attempting to analyze are so complex, the results would be only understandable by experts using the same tool.
- Is it expected that it would take more than 4h to be able to setup the toolchain to reproduce a certain result?

Reproducing and reasoning about results

- When the constructions we are attempting to analyze are so complex, the results would be only understandable by experts using the same tool.
- Is it expected that it would take more than 4h to be able to setup the toolchain to reproduce a certain result?
- In some cases, it is impossible to reason about or reproduce results if they were drafted based on a version of a tool that has been deprecated.

Academic Friction

- Novices are expected to be on par with experts when they would like to have their results peer-reviewed.

Academic Friction

- Novices are expected to be on par with experts when they would like to have their results peer-reviewed.
- Non-reproducible results or flawed proofs that were overlooked in the review process fly under the radar and are rarely retracted.

Academic Friction

- Novices are expected to be on par with experts when they would like to have their results peer-reviewed.
- Non-reproducible results or flawed proofs that were overlooked in the review process fly under the radar and are rarely retracted.
- Some work happens behind closed doors (ex: non-public protocols).

Conclusion

Conclusion

- The tools we have are great! For those who are good at them.
- How can we add more interested people to that set.
- The problem does not lie strictly with the tooling, but how can we reduce the friction from our side?
- I would love to hear your thoughts and critiques.