Who is verifying their cryptographic protocols?

FRIDAY, MAY 7, 2021 Matt Bauer, Mike Dodds

Building secure communication systems requires both secure cryptographic primitives and also secure cryptographic protocols that build messaging schemes on top of those primitives. Well-designed protocols are the backbone of almost all modern digital communication, enabling key exchange, entity authentication, secure channels, and anonymous messaging. On the other hand, improperly designed protocols can render the best cryptography useless, for example, if the protocol inadvertently leaks a secret key.

Because cryptographic protocols are simultaneously critical to our infrastructure and extremely difficult to get right, the formal verification community has spent over two decades developing tools and techniques to validate the correctness of protocol designs. The field has progressed to the point where very few scientific barriers prevent us from formally verifying all of the new protocols we develop. Despite this, we still have a long way to go on the adoption curve. Let's take a look at what protocol verification is happening today and who is doing it. Perhaps more importantly, let's also look at who *isn't* verifying their cryptographic protocols and examine potential solutions for closing the gap.

What is cryptographic protocol verification?

At a most basic level, formal verification techniques aim to give a precise mathematical meaning (or *semantics)* to real-world processes and objects. This semantics captures how these entities behave and provides an environment for verifying, with mathematical precision, what is and is not true of a system. So long as the encoding is accurate, properties proven over the semantics should hold over the real system.

In the case of cryptographic protocols, we seek to prove properties like the absence of man-in-the-middle attacks, data leaks, and privacy violations. When we formally verify a protocol, we establish a resilience to these kinds of flaws against an adversary with a particular set of capabilities. It's impossible to verify protocols correct with respect to adversaries with unlimited power. For instance, an adversary with unbounded computational power could decrypt any message by trying every possible encryption key. So instead, protocols are typically verified under two main types of adversaries with different limitations in their power.

In the *computational model*, correctness is established with respect to a computationally bounded adversary that manipulates bitstrings. Formal verification efforts in this framework involve mechanizing the pen-and-paper proofs typically done by cryptographers. It requires taking an English proof written by a mathematician and encoding the underlying mathematical

objects and inference steps into a machine-readable form that a computer can check. While there is some tooling in this space, including EasyCrypt and CryptoVerif, proofs in the computational model often require extensive manual effort by cryptographic experts.

The verification community has centered around another analysis framework called the *symbolic model to benefit from full automation*. By simplifying the threat model to a Dolev-Yao attacker which has full control of the network but that can only manipulate symbolic messages, verification becomes amenable to automation. You can think of this analysis framework as one that treats the cryptographic primitives as a black box. For a key k and a message m, a ciphertext is represented as a symbolic term like enc(m,k),and such a message can only be decrypted if the attacker holds the secret key k. In spite of the fact that the symbolic model narrows the attack surface, formal verification efforts in this framework are still effective at ruling out classes of attacks and have proven successful in finding bugs in a number of widely used protocols, including Google's single-sign-on protocol. The most well-known symbolic verification tools include ProVerif, Tamarin, and Maude-NPA.

What kinds of protocols are being verified?

There is a great deal of protocol verification happening today. Some of the most exciting work involves verifying new, novel protocols. For example, the Signal protocol combines several sophisticated cryptographic primitives to achieve an array of deep security properties. Its ensuing verification efforts required researchers to meet at the cutting edge of both crypto and verification research. In the computation model, formally verifying these protocols requires extremely complicated and intricate mathematical reasoning to be encoded in machine-checkable format. On the other hand, symbolic tools often won't analyze these complex protocols out of the box, meaning new analysis techniques need to be designed.

Still important, but somewhat less challenging to verify are the protocols developed by standards bodies and major industry frameworks. These protocols are high impact, often use (variants of) well understood cryptographic primitives and are rigorously documented, making them great targets for tool builders and academic security researchers. Perhaps the best example of the verification community rallying around an industrial protocol came with the development of TLS 1.3. Numerous verification efforts using multiple tools took place hand in hand with the development efforts and helped to influence the protocol's design. This paper is a great summary of those results. Major industrial standards such as Bluetooth and 5G have received similar scrutiny from the academic community.

Protocols follow the long tail rule: a few big important protocols are verified, but many more protocols exist in the long tail, and most of these remain unverified. The problem with focusing on major protocols like TLS and Bluetooth is that those protocols are already the subject of

extensive human review and attack. Security researchers and hackers spend years creatively trying to uncover and exploit flaws in their designs. As a result, formal verification of these protocols is less likely to uncover flaws. The majority of bugs are in the long tail, and these are the protocols that aren't getting formally verified. Chances are if a protocol isn't the backbone of an industrial standard or a novel piece of open source cryptography, it only receives manual human review and testing, which is prone to error and oversight. To make matters worse, the teams developing smaller-scale protocols often don't have the in-house cryptographic expertise necessary to perfect their designs. Unlike the case with TLS and Bluetooth, the academic community can't come to the rescue for these smaller scale and proprietary protocols. For every interesting research article on formally verifying aerospace, automotive, IoT, or power grid protocols, there are likely dozens of similar protocols that haven't received the same level of care. To extend the reach of verification in this domain, we really need a grassroots level of adoption where tools are accessible to industrial teams who want to carry out the verification work themselves.

Challenges for adoption

It's common in formal methods research to claim that the complexity of tools is the main barrier to adoption. For example, proving functional correctness of a code block or operating a theorem prover requires a high baseline level of knowledge. I would argue that this is actually not the case for protocol analysis tools. Let's stop to think for a second about what the ingredients are for a symbolic analysis:

1. Specify the protocol
2. Specify the attacker and cryptographic primitives
3. Specify the security properties of interest

In most cases, proprietary protocols aren't starting with a blank slate. They are bending and molding existing protocols to fit a specific use case. If you are building secure messaging infrastructure for IoT devices and your protocol doesn't resemble any well-known schemes, there is probably something wrong. The good news is that this means that steps 1-3 aren't going to require a lot of intensive modeling effort. Equational theories for well known crypto primitives can be pulled off the shelf, and existing protocol and property specifications can be adapted from a slew of representative examples. So why the adoption gap? I would argue that there are two basic hurdles: 1) being able to understand and operate the tools and 2) being able to demonstrate the return on investment compared to the cost of formal verification.

Let's start with the first challenge, using the tools. While the theoretical complexity of tool use is low, adopting any new technology takes energy. Right off the jump, you need to wade through a sea of possible protocol verification tools and approaches, each with its own guarantees,

assumptions, and limitations. Making the right selection can take time. Each tool will have its own specification and modeling language that needs to be well understood before any analysis can occur.

The second challenge is being able to articulate ROI. Cryptographic protocol verification clearly has had a big impact on improving the security of our communication infrastructure as a whole. The nauce comes in articulating that value on a project by project basis. Is there a visible outcome from a symbolic protocol verification that can be presented to budget holders? There is, of course, no guarantee of finding bugs. And most development teams are reasonably confident in their protocol design work, so a reaffirmation of a protocol's correctness may not weigh heavily in planning. Compounding the problem further, the type of vulnerabilities that can be ruled out via formal verification varies from tool and tool. Each has its own reasoning framework and subtle assumptions of the threat model.

Next steps

I believe what's needed is to unify behind a common language and analysis framework for protocols. Every symbolic protocol verification tool is underpinned by the same core logical framework (the applied pi-calculus), yet every tool implements its own variation. Unifying behind a common specification and analysis framework would go a long way towards simplifying the messaging and curbing the ease of adoption and use. When protocol designers write protocols, they typically write them in what's called *security protocol notation* (or Alice and Bob notation). It's a very intuitive way to write protocols that is accessible at the undergraduate level, but it's still rigorous enough to permit deep analysis. Why then, do we require these rigorous designs to be translated into one of a dozen potential languages provided by the analysis tools? We should meet protocol designers where they are, and provide analysis tools that build off the models they already have. This shortens the learning curve and makes it easier to build clear messaging around a common language. Works like, Alice and Bob: Reconciling Formal Models and Implementation, represent important progress in that direction in that they give simple high-level semantics for Alice and Bob notation as well as precise low-level semantics that is amenable to analysis with existing symbolic protocol verification tools.

Formal verification is starting to show its worth in real-world protocol assurance. Verification can provide a high level of confidence in systems or even eliminate certain types of bugs. So far, the big wins in protocol verification have focused on the most widespread types of protocols. I believe it's time we democratize protocol verification and make the highest standards of assurance available to any team that wants it.

If you're interested in discussing cryptographic protocol verification further, please contact us!