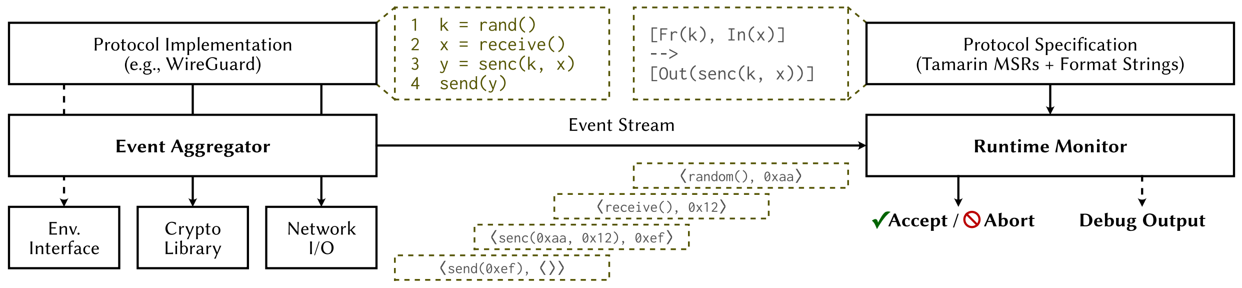
SpecMon: Unifying Verification and Monitoring for WireGuard

Kevin Morio
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
kevin.morio@cispa.de

Robert Künnemann
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
robert.kuennemann@cispa.de



```
k = rand()
x = receive()
y = senc(k, x)
send(y)
```

```
[Fr(k), In(x)]
-->
[Out(senc(k, x))]
```

Fig. 1. SpecMon's architecture, connecting an event aggregator to the monitor.

***Abstract*—Formal verification tools provide strong security guarantees for protocol models, but these guarantees do not transfer to real-world implementations. Runtime monitoring is a promising approach to bridge this gap, yet it introduces a critical new challenge: the specification used for monitoring can diverge from the one used for verification, silently undermining the formal guarantees.**

In this work, we solve this problem by developing a unified specification for the WireGuard VPN protocol. We demonstrate that our unified model is simultaneously suitable for formal verification in Tamarin and runtime monitoring with our tool SpecMon of a Go implementation. This approach not only eliminates specification divergence but also enhances the precision of the original model with minimal changes. Our work establishes that a unified model is a practical and crucial step towards building trustworthy security protocol implementations.

***Index Terms*—Security protocols, Monitoring, Verification, WireGuard, SpecMon, Tamarin**

I. Introduction

The foundation of today's connected world is secure communication. To make communication secure, researchers have created numerous security protocols to meet their different requirements. The past has shown that the design of secure protocols is a challenging task with the risk of leaking sensitive information which could lead to drastic consequences. While proving desired properties of these protocols by hand is theoretically possible, it is prone to human error and, with their increasing complexity, requires significant effort.

To increase the trust in security protocols, researchers have developed verification tools that allow them to formally model a protocol, define the security requirements, and automatically obtain a proof or counterexample for each property.

These tools give us strong formal guarantees of the protocol's model, a necessary first step. However, they provide no guarantees for actual implementations of the protocol—what is executed in the real world and where leaks happen. The reasons for this are manifold: Developers may accidentally use the wrong cryptographic primitives or use them in an insecure way, implement logic flaws, or even maliciously hamper security by introducing a back-channel. There is a gap between the strong results from the formal model and the lack of them for real-world implementations.

While runtime monitoring can bridge this verification gap, it requires a formal specification of the expected behavior. If this monitoring specification is developed independently of the verification model, they can diverge. This divergence is perilous: a property may be proven in the verification model but not be enforced by the monitor, leading to a false sense of security.

We proposed SpecMon [1], a tool that uses Tamarin models [2] for monitoring. In this work, we leverage SpecMon to address the divergence problem by creating a **unified** model for WireGuard that serves both for verification and monitoring.

II. SpecMon Overview

SpecMon's architecture, shown in Figure 1, consists of two mostly independent parts: an event aggregator and the monitor itself.

The event aggregator observes relevant program events of the implementation at runtime and records them in an event trace. The set of relevant program events includes cryptographic operations, network communication, as well as environment events like file access. The event aggregator can be realized with the most suitable technique for the scenario. In a black-box scenario where no source code access is available, an instrumentation toolkit like Frida [3] can be used. If we have access to the crypto and networking library's source code, we can annotate the relevant functions.

TABLE I
Key differences between our unified and the reference model.

Component	**Model** in [4]	**This work**
Message format	Tuples	Format strings
Long-term keys	Fresh names	Read from files
Key derivation	Abstracted	Fully specified
DDoS protect.	Public names	Precomputed
Counters	Public names	Natural numbers
Message exchange	Single message	Multi-message

SpecMon reads a given model file in Tamarin's specification language and turns it into a monitor for a particular agent role (e.g., client or server). It then consumes the event trace from the event aggregator and checks if each event is permissible in the current state. If a violation is detected, SpecMon can terminate the monitored program and notify the user.

III. Case Study: WireGuard

WireGuard [4] is a state-of-the-art VPN protocol with wide adoption. Due to its use of modern cryptography (Noise protocol framework) and a formal Tamarin model co-authored by WireGuard's creator, it serves as an ideal case study for our unified approach. For this, we use the userspace Go implementation [5].

Our primary goal in this case study was to create a single, unified model of WireGuard suitable for both monitoring with SpecMon and verification with Tamarin. This unification ensures that the specification proven correct is the same one used for monitoring, preventing the divergence described in the introduction. This is a significant evolution from our previous work in [1], which focused primarily on the feasibility of monitoring from a Tamarin model.

To create a unified model, we first merged the previously separate multi-set rewriting rules (MSRs) for the initiator and responder roles and re-added actions required for verification. The main challenge in creating a unified model is bridging the abstraction gap between the symbolic world of verification and the concrete world of an implementation. We bridge this gap through careful use of Tamarin's preprocessor to select the appropriate rules and macros. For instance, in verification, new keys are modeled as abstract fresh names. For monitoring, the preprocessor selects a different rule where a concrete key is loaded from a file. Similarly, abstract message tuples for verification are replaced by format-string macros that define the exact byte-level layout for monitoring. This allows a single model source to serve both Tamarin and SpecMon without modification.

The remaining key differences, summarized in Table I, are due to abstractions in the original model, which, for instance, only allowed a single message exchange and abstracted away message counters.

For evaluation, we monitored a WireGuard server connected to 100 clients, with each of them sending a ping before terminating. We also used the recorded event trace for offline monitoring. In the second setting, we monitored one of the clients instead of the server. In both cases, each event is accepted by SpecMon, ensuring that the implementation adheres to the model.

To demonstrate that our approach can also detect deviations from the model, we have deliberately introduced errors into the implementation. For instance, we modified the implementation to reuse ephemeral keys, and in another case, we prevented the transport message counter from being incremented. In both scenarios, the monitor successfully identified the deviation and terminated the program.

The structural changes made to create the unified model necessitated re-validating the original security properties in Tamarin. We successfully re-verified all original lemmas on our more precise model. This process resulted in a modest increase in verification time from approximately 25 seconds to 100 seconds, which we consider an acceptable trade-off for the benefits of unification.

Our analysis revealed that the original model's identity hiding property relied on a verification-specific artifact: a surrogate key added to the encrypted payload. Our unified model, which enforces the implementation's true message format, does not permit such an artificial element at runtime. We addressed this by replacing the surrogate's fresh name with a constant value, which is ignored during monitoring. This approach allowed us to successfully complete the proof while preserving the model's compatibility with the real-world protocol implementation.

IV. Discussion and Future Work

This work demonstrates that unified models for formal verification and runtime monitoring are practical for real-world protocols like WireGuard, resulting in a Tamarin model that is more precise than the original reference. Our approach is designed to uphold the same security guarantees while preventing the critical issue of specification divergence. This unified model provides a reusable template for monitoring any implementation of the protocol, significantly lowering the maintenance burden compared to managing two separate specifications and increasing the trust that can be placed in an implementation's security.

Our work opens several avenues for future research. A key direction is to further automate the creation and refinement of unified models. We plan to explore techniques for inferring message formats directly from observed network traffic, reducing the manual effort required to adapt a symbolic model. Another promising direction is the automated completion of partial models. When SpecMon rejects an event, it already provides a list of permissible next steps; this feedback could be used to suggest new rules to a developer, enabling a semi-automated, interactive model refinement process.

Finally, while SpecMon's performance is already practical, we see opportunities for further optimization, especially for high-throughput applications.

V. Artifact Availability

Our artifact is available on Zenodo [6] and includes all components necessary to reproduce our WireGuard case study:

- The unified Tamarin model for WireGuard that serves both verification and monitoring
- Dockerfiles for running the monitoring evaluation of WireGuard and creating a Tamarin image
- Pre-built Docker containers as tarballs for immediate use
- Comprehensive documentation with reproduction instructions

The evaluation scripts are integrated into the Docker containers. Source code for SpecMon, required libraries, and the WireGuard-Go implementation are fetched from GitHub using fixed commit hashes during container build to ensure reproducibility.

References

[1] K. Morio and R. Künnemann, "SpecMon: Modular Black-Box Runtime Monitoring of Security Protocols," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, in CCS '24. Salt Lake City, UT, USA: Association for Computing Machinery, 2024, pp. 2741–2755. doi: 10.1145/3658644.3690197.

[2] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN Prover for the Symbolic Analysis of Security Protocols," in *Computer Aided Verification*, N. Sharygina and H. Veith, Eds., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 696–701. doi: 10.1007/978-3-642-39799-8_48.

[3] F. Developers, "Frida Instrumentation Toolkit." [Online]. Available: https://frida.re/

[4] J. A. Donenfeld, "WireGuard: Next Generation Kernel Network Tunnel," in *Proceedings 2017 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2017. doi: 10.14722/ndss.2017.23160.

[5] J. A. Donenfeld and contributors, "wireguard-go: A Go implementation of WireGuard." [Online]. Available: https://github.com/WireGuard/wireguard-go

[6] K. Morio and R. Künnemann, "SpecMon: Unifying Verification and Monitoring for WireGuard (RVCase '25 Artifact)." [Online]. Available: https://doi.org/10.5281/zenodo.17023428