Continuous Verification for Cryptographic Protocol Development

Andres Molina-Markham
The MITRE Corporation
a.mm@mitre.org

Paul D. Rowe
The MITRE Corporation
prowe@mitre.org

ABSTRACT

The proliferation of connected devices has motivated a surge in the development of cryptographic protocols to support a diversity of devices and use cases. To address this trend, we propose *continuous verification*, a methodology for secure cryptographic protocol design that consists of three principles: (1) repeated use of verification tools; (2) judicious use of common message components; and (3) inclusion of verifiable model specifications in standards. Our recommendations are derived from previous work in the formal methods community, as well as from our past experiences applying verification tools to improve standards. Through a case study of IETF protocols for the IoT, we illustrate the power of *continuous verification* by (i) discovering flaws in the protocols using the Cryptographic Protocol Shapes Analyzer (CPSA); (ii) identifying the corresponding fixes based on the feedback provided by CPSA; and (iii) demonstrating that verifiable models can be intuitive, concise and suitable for inclusion in standards to enable third-party verification and future modifications.

CCS CONCEPTS

• **Security and privacy → Security protocols**;

KEYWORDS

Cryptographic protocols, verification, IoT

ACM Reference Format:
Andres Molina-Markham and Paul D. Rowe. 2017. Continuous Verification for Cryptographic Protocol Development. In *Proceedings of ACM Workshop on the Internet of Safe Things, Delft, Netherlands, November 5 2017 (SafeThings'17),* 6 pages.
https://doi.org/10.1145/3137003.3137006

1 INTRODUCTION

The Internet Society estimates that 100 billion interconnected devices will be in existence by 2025 [27]. This proliferation of devices with varying constraints and use cases has caused a dramatic increase in the variety of communication patterns to accommodate new relationships among devices. There has been a corresponding increase in the diversity of cryptographic protocols used to secure these communications. Unfortunately, designing cryptographic protocols is challenging because the security guarantees of a cryptographic protocol depend on subtle assumptions.

Typically, designers of protocols rely on expert opinion from individuals in industry and academia to ensure their cryptographic protocols achieve the desired security goals. In the best cases—such as for TLS 1.3—teams of researchers subject a protocol to intense, formal scrutiny before publication and deployment, proving that it meets its goals, or finding and fixing an attack in the process [36]. This requires a good deal of time and attention from protocol design experts. Most protocols for Internet of Things (IoT) are unlikely to be subjected to such expert scrutiny. There are simply not enough experts to scale with the growth of IoT. There is thus a need to help developers design secure IoT protocols on their own.

Fortunately, mechanized verification tools for cryptographic protocols have made great strides in the past few years. Numerous tools are freely available to download and use [12, 14, 29, 32, 37]. These tools effectively mechanize important expertise, making it available to novice users. In this paper, we demonstrate the importance and value of incorporating such tools directly into the protocol development process. Drawing on our own past experience with applying tools to the analysis of proposed standards, and extracting conclusions from the literature on formal methods for cryptographic protocol design, we propose a methodology of *continuous verification* for cryptographic protocol development based on three core principles.

(1) Use verification tools repeatedly while developing the protocol to discover and fix potential attacks.
(2) Common message components are desirable for uniformity, but must be adapted to guard against sub-protocol interactions.
(3) Published specifications should include intuitive but formal models with rigorous semantics that connect to tools.

We illustrate continuous verification with a case study out of the Internet Engineering Task Force (IETF). The reader can find additional supporting analyses at: https://github.com/mitre/cpsa/.

We applied CPSA [37] to *Fluffy: Simple Key Exchange for Constrained Environments* [22], a draft standard which adapts the key management mechanisms of Kerberos [34] into several protocols suitable for IoT. We discovered that several of the protocols did not achieve their goals, which supports our first recommendation.

Supporting our second recommendation, we note that if the authors of the draft had had the feedback provided by CPSA during the earlier stages of development, they would have quickly realized that the use of common message components could introduce attacks. The formal methods community has developed engineering best practices for cryptographic protocols that can help prevent attacks arising from message component reuse [5, 20]. We modified

SafeThings'17, November 5 2017, Delft, Netherlands

ACM ISBN 978-1-4503-5545-2/17/11…$15.00
https://doi.org/10.1145/3137003.3137006

the design of Fluffy to account for more careful reuse of common message components, and verified with CPSA that the protocols could now achieve their security goals.

Our third recommendation is based on the observation that the existence of a verifiable model makes it quite easy to test hypotheses about the effects of alterations to the protocol design. In this way, verification tools may be used in a manner analogous to debuggers for software engineering. As we demonstrate, formal, verifiable models can have intuitive and easy-to-read descriptions. This suggests that such models are indeed suitable for inclusion in published protocol specifications. Their inclusion would allow independent parties to verify security claims. It would also provide a useful starting point for future developers wishing to modify the structures of a protocol to accommodate a different set of constraints.

2 CHALLENGES

There are more than several dozen IoT-related organizations and standards bodies [6], even more if we include organizations such as the Bluetooth Special Interest Group [8], the Zigbee Alliance [45], the Z-Wave Alliance [44], and others. This diversity reflects the heterogeneous nature of the stakeholders and use cases for IoT technologies. This section outlines how this trend impacts the development of cryptographic protocols.

Protocol diversity. In view of the diverse set of applications and stakeholders, it is likely that the number of cryptographic protocols will continue to surge, as we have seen over the last few years [40]. This trend is consistent with one of Saltzer and Kaashoek's system design principles: "Avoid excessive generality" [41]. Reinfurt et al. proposed design patterns for the IoT to address the increasing complexity of IoT deployments [38]. However, the authors recognize that they have identified other candidate patterns and that more will be found over time.

Modification of existing protocols. Some novel cases (e.g., private service discovery and private mutual authentication) require the development of new protocols [43]. However, in many cases, designers develop new cryptographic protocols by performing minor modifications to existing protocols to accommodate special constraints (e.g., connectivity or power consumption). Small changes to cryptographic protocols can significantly impact their security, as we demonstrate in section § 4.3.

Protocols with multiple flows and options. Specifications of cryptographic protocols often describe multiple related protocol flows and options to accommodate diverse scenarios. This practice has two important consequences: (1) *Use of common data structures:* Protocol specifications use common data structures or *building blocks* to facilitate the presentation and implementation of the various protocols. Section § 4.3 gives a concrete example of this use of building blocks; (2) *Detachment of assumptions and protocol flows:* With the objective of being flexible, some protocols specify multiple flows with different security implications. Often, the assumptions for different flows are not explicit. Section § 4.2 describes the security implications in the context of a specific set of protocols.

Rapid development. New protocols are typically motivated by concrete use cases related to specific products that a company would like to bring to market with aggressive development cycles. This implies that protocols, especially those that result from minor modifications to existing protocols, may not always be thoroughly reviewed by cryptographic protocol design experts. This speaks to the need to integrate the use of protocol verification tools throughout the development life-cycle.

3 CONTINUOUS VERIFICATION

Motivated by the trends that we describe in § 2, this section outlines key principles for the development of cryptographic protocol specifications that would result in more secure protocols. These principles are especially relevant when protocol designers do not have access to an in-house team of experts to analyze their protocols. Instead, designers rely on the community at large to help them verify the security of protocols. Concretely, we propose the following principles for the design of cryptographic protocols.

Systematically use verification tools. The idea of using verification tools for the development of cryptographic protocols is not new [3, 7, 37]. However, we argue that cryptographic protocols that are designed for practical use must be formally verified.

Moreover, cryptographic protocol designers should not restrict the use of verification tools to the later stages of the development process. Verification tools should be used at early stages and repeatedly *throughout* the design process. The output of verification tools, such as CPSA [37], offers intuition (usually within seconds) that can help to identify issues and possible solutions. Verification tools can help designers in a similar way to how compilers and debuggers help programmers. Programmers do not use compilers and debuggers only at the later stages of the development process. Increasingly, compilers provide recommendations to programmers to improve code, based on amassed knowledge from a large number of experts over time.

This paper provides several concrete examples of how the use of verification tools helps to identify issues. For example, through the use of CPSA, we identified a problem with the use of the *authenticator*, a message component used in Kerberos [34] and adapted for use in several of Fluffy's protocols [22]. CPSA's visual and intuitive output also hinted at a potential way to fix the issue. A protocol designer can, just as a programmer would, implement a fix to this authenticator, and get immediate verification feedback, as we describe in more detail in § 4.3.

Use common message components judiciously. A common trend is to describe protocols with a variety of flows and options often within the same standards specification. As a result, a family of protocols may be described using messages with common components. For example, the draft of Fluffy [22] describes messages of various protocols using *minitickets, receipts, authenticators and acknowledgements.* These components are combined in messages for various flows in slightly different ways. This practice makes it difficult to reason about the implications of modifying a single component with respect to the security of several protocols.

Relying on the systematic use of verification tools can mitigate the issues. However, we note that we need to perform more research to clearly define a relationship between the goals of message components and the security goals of a cryptographic protocol. We describe this topic in further detail in § 5. Until this relation is well understood, we advocate for the careful use of message components when describing multiple related protocols. In particular, we

recommend minimizing the number of purposes and distinct uses of such common message components. Using a common component in multiple protocols for different purposes entangles such protocols and, therefore, adds complexity. For example, if a message component has a parameter or field which is used differently in multiple flows, this is a sign of dangerous entanglement.

Include verifiable specifications in drafts. It is difficult to use natural language to describe cryptographic protocols. A specification of a cryptographic protocol must be unambiguous. This has motivated the development of multiple domain-specific languages with precise syntax and semantics. Furthermore, some of these languages allow for specifications that can be formally verified [3, 7, 37].

Including verifiable specifications in standards drafts or similar core specification documents can be extremely beneficial [9, 21]. A verifiable specification: (1) provides evidence that a protocol has been subject to formal verification; (2) allows others to validate security analyses; (3) enables others to perform modifications to protocols that they can verify; (4) disambiguates assumptions and formalizes statements that can be inadequately described using natural language.

Standards organizations, such as the IETF, embrace notions related to collective wisdom and specifications with demonstrable utility: *"We reject kings, presidents and voting. We believe in rough consensus and running code"—David Clark [24].*

Other organizations, such as the Open Connectivity Foundation [35], sponsor projects that develop reference implementations [1]. In the case of cryptographic protocols, verifiable specifications are at least as important as "running code." We view the inclusion of protocol models as analogous to reference implementations. While the latter are aimed at those implementing the protocol, the former would be aimed at those evaluating protocol security.

4 PRACTICAL CONTINUOUS VERIFICATION

This section illustrates how *continuous verification* assists the design of cryptographic protocols for the IoT. Concretely, we describe how we can improve *Fluffy: Simplified Key Exchange for Constrained Environments* [22], a set of protocols for the IoT based on Kerberos [34]. We use CPSA for the specification and the verification of the protocols. We discuss how two of the protocols, which may look similar, are substantially different, despite involving similar parties, and sharing common message components or *common building blocks*. The use of one building block—the authenticator—is adequate in one protocol, but it is not adequate in another one.

We describe an attack and a fix, both discovered with the use of CPSA, which is a mature verification tool with strong formal foundations that aims to be simple and intuitive. Finally, we describe how these protocols can be unambiguously described using CPSA S-expressions. These can be formally verified and have intuitive representations that can be complemented with automatically generated diagrams.

4.1 Fluffy: Simplified Key Exchange for Constrained Environments

Fluffy is a set of key exchange protocols that aim to simplify Kerberos for use in constrained environments [22], such as those that Gerdes et al. describe in RFC 7744 [16]. The draft of Fluffy that we analyzed describes five protocols:

- Pair-wise shared key establishment (PSKE)
- Group shared key establishment (GSKE)
- Public key pair establishment (PKPE)
- PSKE key deletion
- GSKE key deletion

The protocols involve devices participating in one of three roles: a *client*, a *service principal* (SP), or a *simple key distribution center* (SKDC). By design, a device can take multiple roles to accommodate situations in which devices have several purposes or are shared by many individuals, for example[1].

The draft of Fluffy describes these protocols using seven common data structures termed *common building blocks*: *SKDC request body*, *miniticket*, *receipt*, *authenticator*, *acknowledgement*, *key data*, *key envelope*. The first five are based on data structures introduced by Kerberos [34], while the last two are new. For example, the miniticket is designed to be functionally equivalent to Kerberos' service ticket; the receipt to be functionally equivalent to Kerberos' SKDC-Response; and, the authenticator to be functionally equivalent to Kerberos' authenticator.

Our goal is to illustrate how the use of the key cryptographic protocol principles that we described in § 3 can help to design more secure protocols. Fluffy suggests several sensible design changes that indeed simplify Kerberos. However, we show it is possible to introduce side effects when modifying an existing protocol.

Pair-wise Shared Key Establishment. The purpose of the PSKE protocol is to enable a client to request from an SKDC the creation of a shared key to communicate securely with a service principal (SP). PSKE's flow involves four messages. In order to simplify the presentation of this paper, we omit some parts of the messages that are irrelevant to the security analysis. For example, we omit optional or informational fields. The readers can find the analysis with all the fields in the draft in our repository.

The client includes an authenticator in its messages to the SKDC and the SP, which cryptographically protects the client's identity and a nonce, in addition to a key shared with the expected recipient. The SKDC's response to the client's request includes a miniticket and a receipt. The miniticket includes the identities of the SKDC, and the SP, together with a cryptographically protected tuple that contains the identity of the client and the newly generated key. This tuple is encrypted with a key shared between the SKDC and the SP. This miniticket is then relayed by the client in message 3. The receipt cryptographically protects a tuple with the key shared between the SKDC and the client. The tuple in the receipt contains the identities of the SKDC, the SP, the nonce generated by the client in the first message, and the requested key. After the SP verifies the message with the miniticket and the second authenticator, it responds with an acknowledgement, which cryptographically protects a tuple with the newly exchanged key. The acknowledgment includes the identities of the client and the SP, and the second nonce generated by the client.

A CPSA analysis of this protocol shows that this protocol does not have security flaws. However, as we discuss in § 4.3, CPSA's output for PSKE and GSKE motivated a change in the authenticator.

[1] RFC 7744 [16] provides rationale for multi-role devices.

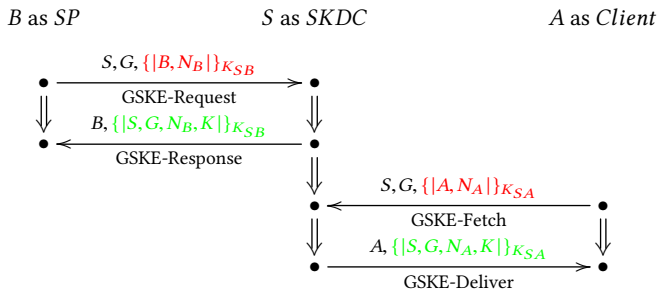

Figure 1: GSKE. B (SP) requests a group key K from S (SKDC). The request includes an authenticator with a nonce N_B. The SKDC responds with a message that contains a receipt with the group key protected with the long-term shared key K_{SB} between S and B. Subsequently, a client A fetches the group key K from the SKDC. The response from the SKDC includes a receipt with the group key, protected with the long-term key K_{SA} shared between S and A. $\{|d|\}_K$ denotes that d is encrypted with K. Vertical arrows $\Rightarrow$ denote immediate causal precedence in the *strand space* formalism [15].

This change simultaneously fixes a security issue in GSKE and also improves PSKE eliminating unsuccessful runs of the protocol that do not result in security vulnerabilities but that would require implementations to handle awkward cases.

Group Shared Key Establishment. GSKE is meant to enable an SP to request a group key from the SKDC. This group key would then be shared with multiple clients to allow for communication with *group authenticity*. As we illustrate in Figure 1, the SP is supposed to request the creation of the group key. After the SKDC creates the group key and shares it with the SP, clients can request a copy of the key. The flows between the SP and the SKDC (the request), and between the client and the SKDC (the fetch), are nearly identical. The difference is in the type of message (e.g. GSKE-Request vs GSKE-Fetch), which is itself part of the message and declared by the sender. The authenticator and the receipt are supposed to ensure the security of the protocol. However, we verified the protocol with CPSA, which identified several *shapes* (possible protocol runs) that highlighted inconsistencies (these can be thought of as "warnings") that often result in possible attacks. In this case, CPSA's output diagrams highlighted several abnormal shapes (e.g., GSKE enables runs where the client's flow precedes the SP's flow).

4.2 Stealing Group Ownership

When an SP requests the generation of a group key, the SP becomes the owner of the group. Group ownership gives an entity important capabilities. Fluffy specifies that only the SKDC and the group owner can request the deletion of the corresponding key. In addition, depending on the authorization policies used in combination with Fluffy, the group owner may control group membership.

Abnormal shapes illustrate an issue with GSKE, namely, that the message types are not protected, and therefore, an attacker

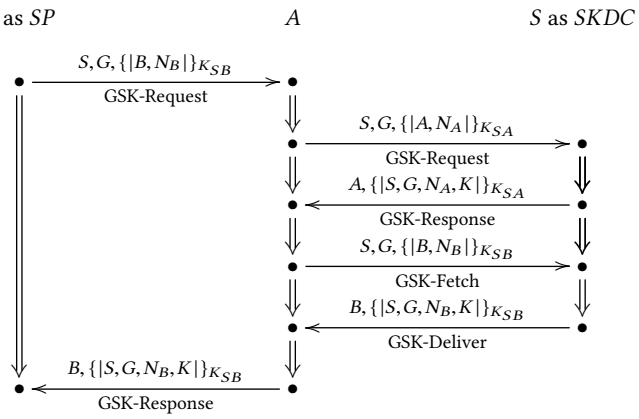

Figure 2: A malicious party A intercepts the request from B, and creates a request to hijack the ownership of the group.

A can intercept and modify the message types to steal the group membership, for example with a flow such as the one in Figure 2.

To illustrate the security implications, consider the following home automation scenario (related to one of the representative use cases described in RFC 7744 [16]): In today's smart homes, a variety of devices are connected through a router or hub. Devices, such as smart light bulbs, wall switches, connected door locks or entertainment systems (e.g., Apple TV or Roku), are all potentially connected via a single wireless router. A conventional approach for restricting communication among devices relies on the ability to ensure group-authenticity, as defined in RFC 3740 [23].

For example, Alice—a home owner—expects that only a subset of her devices control the door lock (i.e., a specific app on her smartphone and a door keypad). On the other hand, Alice also has a $35 screencast device that allows visitors to display content on her living room screen. Alice likes that visitors simply need to "tap" on Alice's hub to establish a link between visitors' smartphones and the screencast device.

Alice's hub implements an SKDC; the door lock and the screencast are service principals, and the door keypads and smartphones are clients for the corresponding services or applications. The authorization policies enforced by Alice's hub are very simple and rely on group membership. In particular, the policies state that (1) only specific door keypads and specific instances of a smartphone app can join groups created by the door lock; on the other hand, (2) anyone can join groups created by Alice's screencast. Alice felt comfortable about adding the inexpensive screencast device to her network because her hub enforces an additional policy: (3) the screencast device cannot join any group that the screencast did not create. Alice (and the designers of her hub) expected that this third policy would prevent a device like the screencast device from communicating with any other device in the home (except for visitors' smartphones). Therefore, they expected to minimize the security impacts of adding a device like the screencast. However, this is not the case. Alice's screencast might be developed by a compromised entity with motivation to allow anyone—without the end user's knowledge—to control anything in a smart home where the

rogue device is added. For example, the screencast could intercept requests from the door lock when it attempts to create a group key. Subsequently, using the flow that we described, the screencast would steal the ownership of the group. Therefore, policy (2) would apply and policy (3) would fail to be enforced.

4.3 Verifiable Specifications

We argue that verification technologies can improve the specification process. Unambiguous and verifiable specifications, such as those used by CPSA [33], can be easy for humans to read. For example, the snippet in Figure 3 illustrates a specification of GSKE that includes a fixed authenticator[2]. Importantly, protocol fixes can be verified by others. For example, a new version of GSKE that applies the fix has exactly one shape from the point of view of each of the three entities. This shape corresponds to the intended run of the protocol. Also, when a family of protocols uses common building blocks, it is important to check that all of the relevant protocols remain secure after a modification. In this case, the fix to the authenticator also reduces the number of shapes of the PSKE protocol[3].

```
(herald "Fixed Group Shared Key Establishment")
(defprotocol fluffy basic
  (defrole sp
    (vars (b s name) (nb g text) (gk skey))
    (trace
      (send (cat "req" s g
        (enc (hash (cat "req" s g)) b nb (ltk s b))))
      (recv (cat "resp" b (enc s g nb gk (ltk s b))))))
  (defrole keyserv
    (vars (a b s name) (nb na g text) (gk skey))
    (trace
      (recv (cat "req" s g
        (enc (hash (cat "req" s g)) b nb (ltk s b))))
      (send (cat "resp" b (enc s g nb gk (ltk s b))))
      (recv (cat "fetch" s g
        (enc (hash (cat "fetch" s g)) a na (ltk s a))))
      (send (cat "deliver" a (enc s g na gk (ltk s a)))))
    (uniq-gen gk))
  (defrole client
    (vars (a b s name) (na g text) (gk skey))
    (trace
      (send (cat "fetch" s g
        (enc (hash (cat "fetch" s g))  a na (ltk s a))))
      (recv (cat "deliver" a (enc s g na gk (ltk s a)))))))
```

Figure 3: Fixed GSKE.

5 CONCLUDING THOUGHTS

We proposed a continuous verification methodology for cryptographic protocol design. Nevertheless, this methodology presents new challenges that may stimulate research.

The first key principle for continuous verification is the systematic use of verification tools. However, it has been difficult to foster adoption of such tools by practitioners who author cryptographic protocol specifications and standards. A more common model has been to elicit analyses before release from academics who are experts in verification methods and tools (see, e.g., [5, 10, 11, 31]). This has typically followed the "reactive" verification model of design-release-break-patch [36], although the development of TLS 1.3 in the IETF has followed a more promising "proactive" verification model of design-break-fix-release. The approach taken for TLS 1.3 may be appropriate for high-profile protocols, but most protocols are not likely to garner the necessary attention from the verification experts who know how to apply the tools.

It is natural to ask why a more continuous verification method has not emerged in practice. We speculate that it stems from three main causes: (1) effective tool use has traditionally required experts in verification, (2) experts' time and focus are commodities in scarce supply, and (3) there are no norms of usage within the governing standards bodies and consortia. There is little to be done about (2). Academics have a high incentive to work on problems with publishable results. There is often little value in publishing the result of an analysis that does not reveal any attacks. As for the other two causes, researchers and practitioners will likely have to work together to address them. We believe it is more effective to train practitioners to use tools than to simply provide them with the results of analysis.

We believe the second main principle of continuous verification—the inclusion of verifiable models in protocol standards—will be an important aspect in establishing norms of use. This principle comes with its own set of challenges. To start, the primary consumers of most standards have traditionally been developers aiming to ensure the results of their development activities can inter-operate with others. A verifiable protocol model does not support this purpose, and so there is likely to be resistance to its inclusion.

The framework described in [30] and standardized by the ISO/IEC in [26] lays the groundwork for including a protocol specification, an adversary model, the claimed security properties against the given adversary, and verification evidence (such as tool outputs) that support the security claims. However, it highlights some important technical challenges. There is no commonly agreed upon description language for protocols that can support rigorous verification. Each tool has its own input language, and while many features of these languages overlap, they differ in subtle ways that might affect the end results. Also, there may be important differences in the specification language of security properties and the underlying adversary models. Research has aimed at addressing cross-tool specification and analysis (e.g. [2, 4, 19, 39]), but more work remains.

Finally, the third principle of continuous verification—judicious use of common building blocks—may be difficult to implement because there are few hard and fast rules for what constitutes "safe" usage of building blocks. By reusing common structures, the dangers of adverse protocol interactions become relevant [28]. One can think of the common building block as a sub-protocol with its own goals that is embedded into several larger protocols. Research has identified design principles that help to ensure a sub-protocol's goals are not subverted by the larger protocol it interacts with [5, 17, 20]. For example, by adding informative data strings to cryptographic units, and by ensuring the same key is never used for two different purposes, many of the pitfalls of protocol interactions can be avoided. Indeed, the weaknesses we discovered in

[2]The analysis of Fluffy suggested the source of the problem. Message types are not tied to identities. This can be easily fixed by adding one field to the authenticator: *a hash of the message, not including the authenticator.*

[3]We refer the reader to our repository that includes a specification of the improved PSKE.

Fluffy could easily be addressed by including in each building block a string identifying which sub-protocol and step it pertains to. A related line of research has investigated safe transformations or refinements of protocols to allow for the adaptation of common structures for new purposes without invalidating the security properties achieved by those structures [13, 18, 25, 42].

6 ACKNOWLEDGEMENTS

We thank Thomas Hardjono and Ned Smith, the authors of the Fluffy draft, for their openness, and commitment to ensuring the security of Fluffy. We also thank Joshua D. Guttman and John D. Ramsdell for their feedback on this work[4].

REFERENCES

[1] 2017. About | IoTivity. (2017). https://www.iotivity.org/about
[2] Omar Almousa, Sebastian Mödersheim, and Luca Viganò. 2015. Alice and Bob: Reconciling Formal Models and Implementation. In *Programming Languages with Applications to Biology and Security - Essays Dedicated to Pierpaolo Degano on the Occasion of His 65th Birthday*. 66–85.
[3] David Basin, Jannik Dreier, and Ralf Sasse. 2015. Automated symbolic proofs of observational equivalence. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1144–1155.
[4] David A. Basin and Cas Cremers. 2014. Know Your Enemy: Compromising Adversaries in Protocol Analysis. *ACM Trans. Inf. Syst. Secur.* 17, 2 (2014), 7:1–7:31.
[5] David A. Basin, Cas Cremers, and Simon Meier. 2013. Provably repairing the ISO/IEC 9798 standard for entity authentication. *Journal of Computer Security* 21, 6 (2013), 817–846.
[6] Joachim Bauernberger. 2016. An incomplete List of Organizations and Alliances for the Internet of Things. (March 2016). https://www.linkedin.com/pulse/incomplete-list-organizations-alliances-internet-joachim-bauernberger
[7] Bruno Blanchet. 2014. Automatic Verification of Security Protocols in the Symbolic Model: the Verifier ProVerif. In *Foundations of Security Analysis and Design VII, FOSAD Tutorial Lectures*, Alessandro Aldini, Javier Lopez, and Fabio Martinelli (Eds.). Lecture Notes in Computer Science, Vol. 8604. Springer, 54–87.
[8] Bluetooth. 2017. Volunteer With The Sig | Bluetooth Technology Website. (2017). https://www.bluetooth.com/membership-working-groups/volunteer-with-the-sig
[9] Jonathan Bowen and Victoria Stavridou. 1993. Safety-critical systems, formal methods and standards. *Software Engineering Journal* 8, 4 (1993), 189–209.
[10] Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, Joe-Kai Tsay, and Christopher Walstad. 2008. Breaking and fixing public-key Kerberos. *Inf. Comput.* 206, 2-4 (2008), 402–424.
[11] Cas Cremers. 2011. Key exchange in IPsec revisited: Formal analysis of IKEv1 and IKEv2. In *Computer Security–ESORICS 2011*. Springer.
[12] Cas Cremers and Sjouke Mauw. 2012. *Operational semantics and verification of security protocols*. Springer Science & Business Media.
[13] Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. 2004. Abstraction and Refinement in Protocol Derivation. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*. 30.
[14] Santiago Escobar, Catherine Meadows, and José Meseguer. 2009. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V*. Springer, 1–50.
[15] THAYER Fábrega, F Javier, Jonathan C Herzog, and Joshua D Guttman. 1999. Strand spaces: Proving security protocols correct. *Journal of computer security* 7, 2-3 (1999), 191–230.
[16] Stefanie Gerdes, Ludwig Seitz, Goran Selander, Mehdi Mani, and Sandeep Kumar. 2016. *Use Cases for Authentication and Authorization in Constrained Environments*. RFC RFC7744. IETF.
[17] Thomas Groß and Sebastian Mödersheim. 2011. Vertical Protocol Composition. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011*. 235–250.
[18] Joshua D. Guttman. 2014. Establishing and preserving protocol security goals. *Journal of Computer Security* 22, 2 (2014), 203–267.
[19] Joshua D. Guttman, John D. Ramsdell, and Paul D. Rowe. 2016. *Cross-Tool Semantics for Protocol Security Goals*. Springer, 32–61. https://doi.org/10.1007/978-3-319-49100-4_2
[20] Joshua D. Guttman and F. Javier Thayer. 2000. Protocol Independence through Disjoint Encryption. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop, CSFW '00, Cambridge, England, UK, July 3-5, 2000*. 24–34.
[21] Anthony Hall. 1990. Seven myths of formal methods. *IEEE software* 7, 5 (1990), 11–19.
[22] T Hardjono and N Smith. 2016. *Fluffy: Simplified Key Exchange for Constrained Environments*. Technical Report hardjono-ace-fluffy-03. https://datatracker.ietf.org/doc/rfc7744/
[23] Thomas Hardjono and Brian Weis. 2004. *The multicast group security architecture*. RFC RFC 3740. IETF.
[24] Paul Hoffman and Susan Harris. 2006. *The Tao of IETF-A Novice's Guide to the Internet Engineering Task Force*. Technical Report.
[25] Mei Lin Hui and Gavin Lowe. 2001. Fault-Preserving Simplifying Transformations for Security Protocols. *Journal of Computer Security* 9, 1/2 (2001), 3–46.
[26] ISO/IEC 2011. *29128: Information Technology - Security techniques — Verification of Cryptographic Protocols*. ISO/IEC.
[27] Karen Rose, Scott Eldridge, and Lyman Chapin. 2017. *The Internet of Things: An Overview, Understanding the Issues and Challenges of a More Connected World*. Technical Report. Internet Society. https://www.internetsociety.org/sites/default/files/ISOC-IoT-Overview-20151221-en.pdf
[28] John Kelsey, Bruce Schneier, and David A. Wagner. 1997. Protocol Interactions and the Chosen Protocol Attack. In *Security Protocols, 5th International Workshop, Paris, France, April 7-9, 1997, Proceedings*. 91–104.
[29] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. 2017. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *IEEE European Symposium on Security and Privacy (EuroS&P)*.
[30] Shin'ichiro Matsuo, Kunihiko Miyazaki, Akira Otsuka, and David A. Basin. 2010. How to Evaluate the Security of Real-Life Cryptographic Protocols? - The Cases of ISO/IEC 29128 and CRYPTREC. In *Financial Cryptography and Data Security, FC 2010 Workshops, RLCPS, WECSR, and WLC 2010, Tenerife, Canary Islands, Spain, January 25-28, 2010, Revised Selected Papers*. 182–194.
[31] Catherine Meadows. 1999. Analysis of the Internet Key Exchange Protocol using the NRL Protocol Analyzer. In *Proceedings, 1999 IEEE Symposium on Security and Privacy*. IEEE CS Press.
[32] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. The TAMARIN prover for the symbolic analysis of security protocols. In *International Conference on Computer Aided Verification*. Springer, 696–701.
[33] MITRE. 2017. cpsa: Cryptographic Protocol Shapes Analyzer. (July 2017). https://github.com/mitre/cpsa original-date: 2017-07-11T18:12:14Z.
[34] C Neuman, T Yu, S Hartman, and K Raeburn. 2005. *The Kerberos network authentication service (V5)*. Standards Track RFC4120. https://www.ietf.org/rfc/rfc4120.txt
[35] OCF. 2017. Open Connectivity Foundation (OCF). (2017). https://openconnectivity.org/
[36] Kenneth G. Paterson and Thyla van der Merwe. 2016. Reactive and Proactive Standardisation of TLS. In *Security Standardisation Research - Third International Conference, SSR 2016, Gaithersburg, MD, USA, December 5-6, 2016, Proceedings*. 160–186.
[37] John D Ramsdell, Joshua D Guttman, Moses D Liskov, and Paul D Rowe. 2009. The CPSA Specification: A Reduction System for Searching for Shapes in Cryptographic Protocols. *The MITRE Corporation* (2009).
[38] Lukas Reinfurt, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Andreas Riegg. 2016. Internet of Things Patterns. In *Proceedings of the 21st European Conference on Pattern Languages of Programs (EuroPlop '16)*. ACM, New York, NY, USA, 5:1–5:21. https://doi.org/10.1145/3011784.3011789
[39] Paul D. Rowe, Joshua D. Guttman, and Moses D. Liskov. 2016. Measuring protocol strength with security goals. *Int. J. Inf. Sec.* 15, 6 (2016), 575–596.
[40] Tara Salman and Raj Jain. 2015. Networking Protocols and Standards for Internet of Things. *Internet of Things and Data Analytics Handbook* (2015), 215–238.
[41] Jerome H Saltzer and M Frans Kaashoek. 2009. *Principles of computer system design: an introduction*. Morgan Kaufmann.
[42] Christoph Sprenger and David A. Basin. 2010. Developing security protocols by refinement. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*. 361–374.
[43] David J. Wu, Ankur Taly, Asim Shankar, and Dan Boneh. 2016. Privacy, Discovery, and Authentication for the Internet of Things. *arXiv:1604.06959 [cs]* (April 2016). http://arxiv.org/abs/1604.06959 arXiv: 1604.06959.
[44] Z-Wave. 2017. - The Internet of Things is powered by Z-Wave. (2017). http://z-wavealliance.org/
[45] Zigbee. 2017. zigbee alliance. (2017). http://www.zigbee.org/

[4] Approved for Public Release; Distribution Unlimited. Case Number 17-2780. Technical data was produced for the U.S. Government under Contract. No. W15P7T-13-C-A802, and is subject to the Rights in Technical Data-Noncommercial Items clause at DFARS 252.227-7013 (FEB 2012).