Automated Analysis of Secure Internet of Things Protocols

Jun Young Kim
UNSW Australia, Data61 CSIRO,
WBS Technology
Sydney, Australia
Jun.kim@unsw.edu.au

Ralph Holz
The University of Sydney
Sydney, Australia
ralph.holz@sydney.edu.au

Wen Hu, Sanjay Jha
UNSW Australia, Data61 CSIRO
Sydney, Australia
sanjay@unsw.edu.au

ABSTRACT

Formal security analysis has proven to be a useful tool for tracking modifications in communication protocols in an *automated* manner, where full security analysis of revisions requires minimum efforts. In this paper, we formally analysed prominent IoT protocols and uncovered many critical challenges in practical IoT settings. We address these challenges by using formal symbolic modelling of such protocols under various adversaries and security goals. Furthermore, this paper extends formal analysis to cryptographic Denial-of-Service (DoS) attacks and demonstrates that a vast majority of IoT protocols are vulnerable to such resource exhaustion attacks. We present a cryptographic DoS attack countermeasure that can be generally used in many IoT protocols. Our study of prominent IoT protocols such as CoAP and MQTT shows the benefits of our approach.

ACM Reference format:
Jun Young Kim, Ralph Holz, and Wen Hu, Sanjay Jha. 2017. Automated Analysis of Secure Internet of Things Protocols. In *Proceedings of ACSAC 2017, Orlando, FL. USA, December 4–8, 2017,* 12 pages.
DOI: 10.1145/3134600.3134624

1 INTRODUCTION

In recent years, a number of Internet of Things (IoT) frameworks and associated security protocols and mechanisms have been proposed to realize the vision of the next-generation ubiquitous Internet. However, security guarantees become even more challenging in the IoT environment due to frequent model/code changes, application-specific adversaries, and emerging threats. The security of IoT systems remains one of the top barriers blocking the success of IoT proliferation [10], as many attacks and vulnerabilities have started to target relatively weak IoT applications and devices. As an example of a particularly interersting case, IoT devices constituted 38% of the victims of a cryptocurrency mining worm [5].

ACSAC 2017, Orlando, FL, USA

DOI: 10.1145/3134600.3134624

The security analysis of IoT protocols remains an open challenge. The main reason for this is that the majority of IoT protocols are prone to frequent change; as the vendors and service providers modify them to respond to market demands or application model changes. Any changes in setting, security model or code require exhaustive security analysis. Another reason is that the security properties of applications vary depending on deployment environment, types of devices, network bandwidth, and energy availability [34]. This results in the need for analysis of the same protocol under various adversary models and goals. For instance, some applications consider only the traditional Dolev-Yao (D-Y) attacker (insecure wireless channel), while others must consider a significantly stronger adversary such as the extended Canetti-Krawczyk (eCK) model [36], where the adversary may dynamically compromise a limited number of long-term and session keys with the possibility of corrupting random number generators. This is often due to the unguarded deployment environment of IoT applications. Some applications may also require Perfect Forward Secrecy (PFS), which protects past sessions against future compromises of secret keys [33]. Thus, a full security analysis of ever-changing IoT protocols under application-specific settings is a difficult task.

Pen and paper analysis. One well-established practice for security analysis is 'pen and paper' cryptographic analysis. Despite its clear value, it is error-prone and time-consuming [26], especially for protocol drafts where the security model and implementation details can still change. Even some parameter changes in source code can trigger a substantial full security analysis.

Automated formal symbolic security analysis. Tool-based formal security analysis has helped with *'automated'* analysis of abstract security protocols. In the past decade [21, 35, 39], a large number of formal security analysis tools have become publicly available. One highly interesting case, where such tools have proven to be immensely useful, is the analysis of TLS 1.3 revisions. Given the complicated variants and the use case of TLS 1.2, many flaws such as BEAST [27] and Lucky-Thirteen [20] had been identified before TLS 1.3. Thus, the TLS working group decided to adopt an 'analysis-before-deployment' paradigm when drafting TLS 1.3. Yet the full security analysis of the entire TLS 1.3 draft was not expected to finish in time due to the substantial complexity of the protocol suite, with multiple 'pen and paper' or tool-based approaches. Cremers et al. [26] performed extensive symbolic security analysis on TLS 1.3 draft-10 to confirm that it is robust, except for a design flaw in a delayed client authentication mechanism when combined with a PSK-resumption

handshake. The authors claim that their approach can analyse future TLS 1.3 drafts with a minimum time effort, and they keep track of current drafts.

Challenges. Based on our experience, we found that formal security analysis is extremely useful for standard-based IoT protocols. For instance, standard security primitives such as Public Key Cryptography (PKC) and Pre-Shared Key (PSK) based protocols can be analysed and tracked with minimum effort. However, we discovered several critical limitations while introducing new algorithms.

DoS attack vulnerability. The 2016 DYN cyberattack [12] and 2013 Spamhaus attack [3] have demonstrated that Denial-of-Service (DoS) attacks can cause massive disruption, especially in IoT deployments. IoT systems are particularly vulnerable to DoS attacks due to the involved Machine-to-Machine (M2M) communication. Although the latter enables intelligent applications, its fault-tolerance behaviour and lack of human monitoring brings new vulnerabilities. In addition, when IoT applications run on battery-powered devices, bandwidth and energy are at a premium to maximize the lifetime of such devices.

Contributions. We make the following contributions:

- We perform symbolic security analysis of prominent IoT protocols under various adversary models and goals such as D-Y, eCK, and PFS. We present our analysis and make the code public so that it can be used adaptively upon modification of these protocols[1].
- We present critical challenges in existing symbolic security analysis tools and propose solutions.
- We further demonstrate how DoS attacks can be modelled and protocols correspondingly verified; we show that important IoT protocols are vulnerable to DoS attacks.
- We discuss the limitations of existing DoS countermeasures in IoT and propose a tailored DoS attack countermeasure that can be generally used in other IoT protocols.

This paper is organized as follows. Section 2 presents background of prominent IoT protocols and existing DoS countermeasures. Section 3 introduces the Tamarin prover by modelling standard-based protocols. Section 4 presents restrictions of existing symbolic tools and proposes solutions to address the restrictions. Section 5 presents the innate DoS attack vulnerability of IoT protocols. To address DoS attack vulnerability, we propose our countermeasure in Section 6. Section 7 concludes the paper.

2 RELATED WORK

In this section, we present prominent IoT protocols and their security goals, which we will refer to throughout the paper. We present a summary in Table 1. We also show the existing DoS countermeasures and their shortcomings when they are used in the IoT.

[1]Our implementation is available at https://github.com/jun-kim/Automated-security-verification-of-IoT-protocols

Protocol	Attacker/Goal	Note
Sigfox [14]	D-Y/Standard	All packets carry signature
LoRa [13]	D-Y/Secrecy, Integrity	PSK is recommended
MQTT [15]	D-Y, eCK/Standard	Recommend session resumption
CoAP[4]	D-Y, eCK/Standard	Rely on DTLS
JPAKE [6]	D-Y, eCK/PFS	Eliminate PKC suite

Table 1: Overview of IoT protocols analyzed in this paper.

2.1 LPWAN Protocols

Low Power Wide Area Network (LPWAN) can provide coverage up to several kilometres using low-power applications [13]. Many LPWAN IoT device management applications such as water/gas meters, street lights, vending machines, devices for pets, trash containers, and smoke alarms are potential users of this technology. The LPWAN radio layer uses the unlicensed spectrum below 1 GHz, which is free for use for all applications.

SigFox. SigFox [14] is designed for enterprise use cases with tight management schemes. Since it targets standard security, SigFox mandates a PKC cipher suite with X.509 certificates even on extremely constrained devices. A distinguishable difference between SigFox and standard PKC is that all SigFox communication packets carry the sender's signature to enhance integrity in low-bandwidth environments. Furthermore, all SigFox devices are equipped with manufacturer-generated public and private key pair before deployment and this is used as their identity during their lifetime. Although this design choice enables standard security with tight device management features, performance and battery life of constrained devices are extremely limited. Therefore, SigFox is a favourable choice for enterprise IoT applications where devices have no energy constraints.

LoRa alliance. Designed for long battery life, the Long Range alliance (LoRa) [13] specification essentially uses Pre-Shared Key (PSK) cipher suites with HMAC support only. It leaves the choice of cipher suites to application designers and developers. Compared to SigFox, the LoRa security model is more suitable for constrained IoT applications with long-term usage. However, PSK is only secure against a D-Y adversary; one compromised device can jeopardize the entire application security.

2.2 Publish/Subscribe Protocols

A number of publish/subscribe messaging protocols are currently in use in commercial IoT applications. The Constrained Application Protocol (CoAP) [4] and MQ Telemetry Transport (MQTT) [15] are two prominent examples. They are extremely simple with RESTful interfaces and lightweight publish/subscribe messaging protocols, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. Their design principles strive to make them ideal for emerging IoT applications, where bandwidth and battery power are at a premium. When the server (broker) receives a published item, it securely distributes the item to all subscribers resulting in tight security and privacy provided by

the centralized server. Our case study will model and analyze CoAP and MQTT.

2.3 JPAKE Algorithm

Password-Authenticated Key Exchange (PAKE) schemes allow to establish secure communication between two remote parties based solely on their shared low-entropy password. The PAKE paradigm is suitable for IoT applications; it features straightforward structure, efficiency, and has no PKC infrastructure requirements. As a factory default, IoT devices are only required to be equipped with a low-entropy password that can be used for further secure communication and authentication without relying on complex PKC infrastructure.

Although patent issues have blocked PAKE adoption[2], Password-Authenticated Key Exchange with juggling (JPAKE) [29] has been proposed patent-free. The 'juggling' (hence 'J'PAKE) technique and Schnorr signature [43] for Zero-Knowledge-Proofs (ZKP) enable this design. JPAKE is included in many IoT protocols such as Google Nest's THREAD IoT commissioning protocol[3] [7]. It is currently being standardized by the IETF [6].

2.4 Summary of IoT Protocols

To elaborate on the IoT protocols summarized in Table 1, MQTT, CoAP and SigFox are used as typical examples of standard TLS 1.2 with simple variations depending on the protocol design philosophy. SigFox uses signatures for every packet and MQTT uses a session resumption approach. CoAP, on the other hand, adopts Datagram TLS (DTLS) for low-power networks. Throughout this paper, we will use SigFox as a standard TLS 1.2 example and model MQTT and CoAP in terms of their variations. LoRa will serve as a typical PSK example. Finally, JPAKE will serve as our example to introduce the challenges in formal security analysis of practical IoT protocols.

2.5 DoS Attack Countermeasures

We summarize existing DoS countermeasures for IoT and their shortcomings as follows.

One-way hash. One-way hash [44] have been used in wireless sensor networks (WSNs) as a DoS attack defensive measure. Devices are equipped with a non-invertible hash function (one-way) before deployment and use it for light-weight verification before performing heavy crypto operations. However, this mechanism raises concerns in the IoT domain since equipping a unique function per application from the factory is hard. Critically, a successful node compromise attack can reveal the one-way function, resulting in neutralization of the defence.

Cookie: IKEv2 [1] and DTLS [8] adopt the cookie approach as a DoS attack defensive measure. In DTLS, upon arrival of a client hello message, the server issues a cookie verification message to confirm that the hello is from the specific client. The cookie verification message contains a stateless cookie so the server does not need to keep any record. Despite its effective defence against spoofed IP addresses, this mechanism provides no defence against DoS attacks mounted from valid IP addresses.

Time lock puzzle. Rivest, Shamir, and Wagner (RSW) [40] proposed a time-lock puzzle based on RSA. In the RSW construction, solving a secret value $b = a^{2^l} \, mod \, n$ is required, where a, b are random values in a group G. The parameter l governs the hardness of the puzzle; a solver must perform l modular squaring operations in order to compute b. RSW constructs a kind of digital time capsule which the solver must consume a certain amount of time as intended by the issuer. Even though the RSW approach is a good countermeasure, the issuer must perform one square exponentiation and one multiplication to create the puzzle; these computationally intensive operations can drain the battery on constrained devices.

Client Puzzle. Client puzzles as i [31] are proposed as a cryptographic countermeasure against connection depletion attacks such as TCP SYN flooding. A client puzzle is a quickly computable cryptographic problem making use of a server secret, the time, and the client request. In order to proceed to the next step of the protocol, the requester must solve and submit the answer. The client puzzle approach has multi-fold benefits against DoS attacks, as its generation is lightweight and hardness control is possible. However, it has a linear communication overhead for hardness control due to its sub-puzzle construction algorithm.

We further elaborate on the DoS vulnerability of IoT with our proposed DoS countermeasure in Section 6.

3 SYMBOLIC MODELLING OF STANDARD-BASED PROTOCOLS

We investigated various state-of-the-art verification tools to find the most suitable tool. We found that some tools are limited in modelling complex IoT scenarios. The majority of tools are based on bounded verification, where only a finite subset of behaviours is considered. Unbounded verification tools such as ProVerif [25] are proven efficient, but this does not guarantee termination in the case of complex protocol verifications. Several other tools are restricted to analysis under the traditional D-Y adversary model [38, 39]. In terms of expressivity, some tools are limited to Diffie-Hellman (DH) inverses [25] or exponentiations [35, 37]. Based on our investigation, we believe the Tamarin prover [41] can model IoT protocols without the aforementioned issues. The Tamarin prover has already proven its value in complex protocol analysis and demonstrated that little effort is needed in tracking design changes in the TLS 1.3 specification. Unlike existing tools, Tamarin guarantees the termination of analysis of complex protocols; it also supports DH inverses and exponentiations. Furthermore, various extensions for Tamarin exist, such as bilinear pairing/AC-operator support [42] for group

[2]Many PAKEs are patented (e.g., EKE [24] by Lucent technologies and SPEKE [30] by Phoenix technologies).
[3]The THREAD consortium is organized by Google Nest and other major IT companies such as ARM and Samsung. THREAD released its implementation as open source in May, 2016.

key schemes, human error modelling [23], and observational equivalence modelling [22]. These features will be beneficial for modelling practical IoT application use cases.

In this section, we model standard-based IoT protocols using the Tamarin prover. Manuals and source code for Tamarin can be found on the official page [11].

3.1 Modelling SigFox (PKC)

SigFox essentially uses TLS 1.2 PKC and shares its security goals. The main difference between SigFox and TLS 1.2 PKC is that all packets carry the sender's signature for integrity under low bandwidth. Let us take a simple SigFox notification example below to show how modelling works in Tamarin. The SigFox server (Alice) pushes an asymmetric-encrypted notification (na) with its signature to devices (Bob).

Facts. Tamarin defines a transition system of facts using multiset-rewriting rules. Two types of facts represent properties and resources of the protocol; they can be consumed by rules. Linear facts are used for limited resources so they are consumed only a limited number of times. Persistent facts are defined with an exclamation point; they can be consumed an unlimited number of times by rules.

```
[Premises, Consume facts]--[Actions]->[Conclusions]
                    OR
[Premises, Consume facts]-->[Conclusions]
```

Figure 1: A Tamarin rule consists of 3 sides. The middle (actions) is not always needed.

Rules. Each Tamarin rule has three 'sides': the left side for premises and consuming facts, the middle for defining actions, and the right side for conclusions (see Fig. 1).

In rule Register_pk, the left side rule [Fr(∼ltk)] defines a premise that ∼ltk is a new linear fact using a pre-defined rule Fr() (line 1). The fact ltk uses ∼ to represent a fresh value; it can be consumed in the conclusion rule [!Ltk($A, ∼ltk), !Pk($A, pk(∼ltk), Out(pk(∼ltk)]. The conclusion rule uses two persistent facts. The !Ltk() fact allots a long-term private key to a public ID A ($ means public resource). The !Pk() fact allots a public key to a public ID A. The Out() fact represents transmitting the public key (pk()) to the untrusted network, resulting in the receiver's and adversary's knowledge (line 2). Facts transmitted using Out() can be received by the In() fact; this mechanism represents the traditional D-Y adversary model.

We define the rule Push_notification to show the server's role for transmitting encrypted notification to devices. Public ID A uses the public key of B (pkB) and generates a fresh data item ∼na. It then encrypts the data and A's ID using the pre-defined primitive fact aenc() with pkB to generate the message (msg, line 1). Using another fact sign(), A signs the message using its own private key (ltkA, line 2), which is retrieved by a persistent fact !Ltk(A,ltkA) (line 3). To make multiple occurring terms simple, let-in binding is supported in Tamarin and concatenation of facts is

```
rule Register_pk:
1.[ Fr(~ltkA) ] -->
2.[ !Ltk($A,~ltkA), !Pk($A,pk(~ltkA)), Out(pk(~ltkA))]
```

```
rule Push_notification:
1.let msg = aenc(<A, na>, pkB)
2. sig = sign(msg,ltkA) in
3.[Fr(~na), !Ltk(A, ltkA), !Pk(B, pkB)]--
4.[Send(A, msg), Secret( na), Role('A'), Honest(B)]->
[Out(<msg, sig>)]
```

```
rule Receive:
1.let msg = aenc(<A, na>, pkB) in
2.[!Ltk(B,ltkB),!Pk(A, pkA), In(<msg, sig>)]--
3.[Eq(verify(sig,msg,pkA),true),Recv(B, msg),Secret(na),
Honest(B), Honest(A), Role('B')]->
4.[St_B_1(B, ltkB, pkA, A, na)]
```

```
rule Reveal_ltk:
1.[!Ltk(A, ltkA) ]–[Reveal(A)]-> [ Out(ltkA)]
```

Figure 2: SigFox implementation using Tamarin. This is a typical PKC scenario.

represented as <A, ∼na> (line 1-2). Unlike the rule Register_pk, Push_notification has a middle side [Send(A, msg), Secret(∼na), Role('A'), Honest(B)] (line 4). These action facts represent a transition of states that can be used in the security proof stage.

We model a device's role in rule Receive. B receives the asymmetrically encrypted message with a signature using the In(<msg, sig>) fact (line 2). Tamarin supports <x,y> as syntactic sugar for concatenation of x and y and <x1,x2,...,xn-1,xn> as syntactic sugar for <x1,<x2,..,<xn-1,xn>...>.

B first performs signature verification using the verify() fact and asymmetric decryption using the fact adec() and B's private key (ltkB, line 3). Tamarin provides various axioms such as Eq() for equality, Neq() for inequality, and Unique() for unique actions. Eq(verify(sig,msg,pkA),true) means the rule Receive will proceed provided the signature verification equals true (see Fig. 3). The St_B_1() fact is a state fact to store current state that will be used in later rules (line 4).

To generate a stronger adversary model eCK and PFS, we add a rule Reveal_ltk, where adversaries can have access to long-term private keys. The action fact Reveal(A) will be used to generate contradictions for security properties. This will be used to introduce stronger security goals and adversaries such as eCK and PFS.

Cryptographic primitives. Tamarin provides various pre-defined cryptographic primitives under a *perfect cryptography model*, which means all cryptographic primitives used in Tamarin are *perfect*. For example, symmetric cryptography never reveals plain text, hashing acts as a random oracle, MACs and signatures are unforgeable. Some examples of the supported primitives are described in Fig. 3.

Security properties. Tamarin defines lemmas for properties using its first-order logic expressions. For instance, &

Tamarin Cryptographic Primitives
Diffie-hellman:xˆyˆz = xˆ(y*z), x*inv(x) = 1
Symmetric cryptography: sdec(senc(m,k),k) = m
Asymmetric cryptography: adec(aenc(m,pk(sk)),sk)=m
Digital signature: verify(sign(m,sk),m,pk(sk)) = true
Bilinear-pairing: em(pmult(x,p),q) = pmult(x,em(q,p))
axiom **Equality**: "All x y #i. Eq(x,y) @i ==> x = y"

Figure 3: Tamarin prover provides various built-in primitives under perfect cryptography model.

is for and, | for or; the Ex and All quantifiers mean the usual first-order expressions. Tamarin supports the indicators @ for a point in time and # for a specific variable. Tamarin's property specification language is a guarded fragment of a many-sorted first-order logic with a sort for points in time. This logic supports quantification over both messages and time [11].

Our SigFox lemma definitions are described in Fig. 4.

lemma **executable**: exists-trace
"Ex A B m #i #j. Send(A,m)@i & Recv(B,m)@j"
lemma **secret_A**: all-traces
"All n #i. Secret(n) @i & Role('A') @i ==> (not (Ex #j. K(n)@j)) |(Ex B #j. Reveal(B)@j & Honest(B)@i)"
lemma **secret_B**: all-traces
"All n #i. Secret(n) @i & Role('B') @i ==> (not (Ex #j. K(n)@j)) |(Ex B #j. Reveal(B)@j & Honest(B)@i)"
lemma **secrecy_PFS_A**:
"not All x #i. Secret(x) @i & Role('A') @i ==> not (Ex #j. K(x)@j)|(Ex B #r. Reveal(B)@r & Honest(B) @i & r < i)"

Figure 4: Security properties are defined as lemmas.

To ensure that other lemmas do not just hold vacuously because the model is not executable, we first define a sanity check lemma that shows that the model can run to completion. This is given as a regular lemma, but with the exists-trace keyword, as seen in the lemma 'executable'. This keyword says that the lemma is true if there exists a trace on which the formula holds. The executable lemma states that at times @i and @j, there exists A/B and message m. To verify secrecy claims, we use the Secret(x) action fact to indicate that the message x is supposed to be secret. In our model, both agents may claim secrecy of a message na, but only A's claim is true. To distinguish between the two claims, we added the action facts Role('A') and Role('B') in the rules Push_notification and Receive. We call an agent whose keys are not compromised an honest agent and label it Honest(A) and Honest (B).

Lemma secret_A states that whenever a secret action Secret(n) occurs at time i of Role('A'), the adversary does not know x or an agent claimed to be honest at time i has been compromised at a time r. The security claim of B is defined in lemma secret_B, but only A's claim holds.

We further model the stronger secrecy property PFS, which requires that messages labeled with a secret() action before a compromise remain secret. Although SigFox does not consider this security goal, some applications with possible key disclosure can use this lemma. If the perfect forward secrecy property is negated, PFS does not hold.

The security analysis for SiFox is given in Fig. 5; Tamarin also provides a graphical UI for convenience.

analyzed: sigfox_PKC.spthy
executable (exists-trace): verified (8 steps)
secret_A (all-traces): verified (11 steps)
secret_B (all-traces): falsified - found trace (10 steps)
secrecy_PFS_A (all-traces): falsified - found trace (3 steps)

Figure 5: The result of the SigFox push notification protocol. It can be viewed in a GUI.

3.2 Modelling LoRa (PSK)

We model LoRa PSK scenarios, where A and B exchange their own secret secA and secB, respectively, using symmetric encryption with a HMAC for integrity. We then verify the security under D-Y and eCK adversary models.

First, PSK distribution and revealing are defined in rules Key_distribution and Reveal_psk. In rule A_Send, Alice retrieves her shared key (KeyA) using the !PSK() fact (line 2). She transfers the message with HMAC to the network (line 4). In rule Role_B, Bob receives the message using the In() fact, then retrieves his shared key (keyB, line 3). Bob performs HMAC verification using his own key using the Eq() axiom (line 4). After generating his own secret (secB), he encrypts and sends a message along with a HMAC (line 5). Bob inserted a unique identifier 'B_1' in line 2 to distinguish his message from Alice's message. In rule A_Receive, Alice receives Bob's message using the In() fact (line 2), then verifies the HMAC using the Eq() axiom.

We first verify the correctness of the protocol flow with lemma protocol_flow. It verifies that Alice performs Send_A at time @i and Bob receives and sends back at time @j. Alice receives Bob's secret at time @k. To verify the secrecy of two secret messages secA and secB, we use the action fact Secret(). Lemma message_secret_DY confirms that the two secrets are secure when no-one reveals the PSK. When there is a possibility of key disclosure, the message secrecy is verified as insecure under the eCK model, which is defined in lemma message_secret_eCK.

4 RESTRICTIONS OF SYMBOLIC SECURITY ANALYSIS

Although symbolic analysis can model and track standard-based protocols efficiently, introducing new algorithms is a challenging task. We present our findings in modelling new algorithms such as JPAKE. Analysing new algorithms such as JPAKE is necessary since it has been included in many IoT

```
rule Register_pk:
1.[ Fr(~ltkA) ] -->
2.[ !Ltk($A,~ltkA), !Pk($A,pk(~ltkA)), Out(pk(~ltkA))]

rule Key_distribution:
1.[ Fr(~key)]-->[ !PSK($A, ~key)]

rule Reveal_psk:
1.[!PSK(A,key)]--[Reveal(A)]->[Out(key)]

rule A_Send:
1.let msgA = <A,senc(~secA, keyA)> in
2.[ !PSK(A, keyA), Fr(~secA)]
3.--[Send_A(A,msgA)]->
4.[Out(<msgA, hmac(msgA,keyA)>)]

rule Role_B:
1.let msgA = <A,senc(secA, keyA)>
2. msgB = <B,'B_1',senc(~secB,keyB)> in
3.[!PSK(B,keyB),In(<msgA,
hmac(msgA,keyA)>),Fr(~secB)]
4.--[ Recv_B(B, msgA), Secret(secA), Send_B(B, msgB),
Eq(hmac(msgA,keyA), hmac(msgA,keyB))]->
5.[Out(<msgB,hmac(msgB,keyB)>)]

rule A_Receive:
1.let msgB = <B,'B_1',senc(secB,keyB)> in
2.[ In(<msgB,hmac(msgB,keyB)>), !PSK(A, keyA)]
3.--[Recv_A(A,msgB), Secret(secB), Eq(hmac(msgB,keyB),
hmac(msgB,keyA))]->[]

lemma protocol_flow: exists-trace
"Ex A B SA SB #i #j #k. Send_A(A,SA)@i &
Recv_B(B,SA) @j & Send_B(B,SB)@j & Recv_A(A,SB)@k
& i < j & j < k"
lemma message_secret_DY: all-traces
"All s #i. Secret(s) @i & not (Ex A #i. Reveal(A)@i) ==>
not (Ex #j. K(s)@j)"
lemma message_secret_eCK: all-traces
"All s #i. Secret(s) @i ==> not (Ex #j. K(s)@j)"
```

Figure 6: Tamarin implementation of LoRa alliance PSK with security properties.

protocols such as Google Nest's THREAD commissioning protocol. The (D)TLS handshake is currently under development for use in the IoT [2, 7, 8], too.

We briefly explain the JPAKE protocol. Two parties Alice and Bob first agree on $g \in G \in Z_p$. They share a low-entropy password $s \in [1, q-1]$. Alice generates two ephemeral values $x_1, x_2 \in Z_q$, then transfers g^{x_1}, g^{x_3} with a ZKP for x_1, x_2. Similarly, Bob generates two ephemeral values $x_3, x_4 \in Z_q$, then transfers g^{x_3}, g^{x_4} with a ZKP for x_3, x_4. After verifying the ZKP, Alice transfers $\alpha = g^{(x_1+x_3+x_4)\cdot x_2 \cdot s}$ with ZKP for $x_2 \cdot s$. Similarly, Bob transfers $\beta = g^{(x_1+x_2+x_3)\cdot x_4 \cdot s}$ with a ZKP for $x_4 \cdot s$. After verifying the ZKP, Alice computes $K_A = (\beta / g^{x_2 \cdot x_4 \cdot s})^{x_2} = g^{(x_1+x_3)\cdot x_2 \cdot x_4 \cdot s}$. Similarly Bob computes $K_B = (\alpha / g^{x_2 \cdot x_4 \cdot s})^{x_4} = g^{(x_1+x_3)\cdot x_2 \cdot x_4 \cdot s}$. They establish the same key using a hash function h, as $h(K_A) = h(K_B)$.

Protocol	Security	Result
JPAKE	D-Y	proof
JPAKE	eCK	proof
JPAKE	PFS	proof
CoAP-PSK	D-Y	proof
CoAP-PSK	eCK	attack on message secrecy
CoAP-PKC	eCK	proof
CoAP-PKC	PFS	attack on key establishment
MQTT	D-Y	proof
MQTT	eCK	proof
Sigfox	D-Y	proof
Sigfox	PFS	attack on key establishment
LoRa	D-Y	proof
LoRa	eCK	attack on message secrecy

Table 2: Overview of our case studies. For all protocols, authentication, key establishment, and message secrecy are proven secure under the D-Y model.

When modelling protocols using JPAKE, one encounters a critical problem.

4.1 Unification Problem

Symbolic security analysis of JPAKE is not a trivial task for existing formal security tools since they cannot support the multiplication and addition of DH groups [28]. This is known as the *unification problem*, where multiplication and addition of DH group cause an NP-complete decidability problem.

Let us consider how unification works in symbolic analysis tools. To prove the claim that the adversary never learns the term x, symbolic analysis tools assume the adversary did learn the term and then attempt to find a contradiction. This is done by searching backwards from the adversary learning the term and seeing what rules from the protocol (or the adversary) rule set could have been applied to get there. The primary question we want to investigate is whether two terms can be unified, and if so, what substitutions will unify them. If we want to unify $xy = 2z$ where x,y,z are variables and we are operating in Z_p, the cyclic group of order p, then it is proven that there are at most finitely many unifiers (substitutions that make the two terms identical) such as $(x = 2, y = z)$, $(x = 1, y = 2z)$, $(z = 2^{-1}, x = y^{-1})$. However, the equality $(g^a g^b)^c = (g^a)^c (g^b)^c$ or $g^a g^b = g^{(a+b)}$ cannot be unified [41]. There are infinitely many unifiers for DH groups and finding such unifiers reduces down to the problem of solving Diophantine Equations, which is known to be undecidable [32]. This is a general open problem and research topic in all symbolic verification tools based on unification.

To model the JPAKE protocol correctly, it requires multiplication and addition of DH groups such as $g^{x1} g^{x3} = g^{(x1+x3)}$, as Alice generates her key as $g^{(x1+x3)\cdot x4 \cdot x2 \cdot s}$. Since the unification problem is NP-complete, there is no known efficient algorithm to support the required features. This means some parts of JPAKE such as $g^{(x1+x3)}$ and Schnorr ZKP cannot be included in the model since they require multiplication or addition of DH groups. Although using the

symbolic analysis approach for emerging IoT protocols is useful, introducing new algorithms with multiplication/addition of DH group such as JPAKE is not possible.

To model JPAKE and protocol suites that use it, we propose two approaches. Note that our approaches are not a theoretical contribution to this open research problem, but rather a practical approach to imitate the required functionality.

rule **Password_distribution**:
1.[Fr(pass)]-->
2.[!Alice($A,~pass), !Bob($B,~pass)]

rule **RoleA_1**:
1.let Y1 = 'g' ^~x1
2. Y2 = 'g' ^~x2 in
3.[!Alice(A,pass), Fr(~x1), Fr(~x2)]-->
4.[Out(< *A*,'A_1', Y1, Y2>), ST1_A(A,pass,~x2)]

rule **RoleB_1**:
1.let Y3 = 'g' ^~x3
2. Y4 = 'g' ^~x4
3. KB = h(<Y1 ^~x3, Y2 ^~x4 ^pass>) in
4.[!Bob(B,pass), Fr(~x3),Fr(~x4),In(<A, 'A_1', Y1, Y2>)]
5. --[Sym_Key_B(B,KB)]->
6.[Out(<B,'B_1',Y3, Y4>)]

rule **RoleA_2**:
1.let KA=h(<Y3 ^x1 ,Y4 ^x2 ^pass>) in
2.[In(<B, 'B_1', Y3, Y4>), ST1_A(A,pass,x2)]
3.--[Sym_Key_A(A,KA)]-> []

lemma **key_establishment**: exists-trace
"Ex A B key #i #j. Sym_Key_B(B, key) @i & Sym_Key_A(A,key) @j & i < j"

Figure 7: Approximation of the JPAKE protocol for use as a building block in a larger protocol suite.

Approximation. One solution is to model JPAKE as closely as possible to its original algorithm. This may not capture some properties in the design, but important security properties remain as intended. As explained, we cannot exactly model $g^{(x1+x3)}$ due to the unification problem. As Abdalla et al. analysed [19], the juggling technique in $g^{(x1+x3)}$ is employed to eliminate some randomization factors for on/off-line dictionary resistance, where adversaries can verify the password only once per session. Since symbolic security analysis cannot verify on/off-line dictionary resistance, juggling technique over $g^{(x1+x3)}$ is impossible to model anyway. Thus, our approximation uses all parameters, but uses concatenation ($\|$) instead of exponentiation over the parameters. In our approximation, Alice generates her key as ($g^{x1 \cdot x3} \parallel g^{x4 \cdot x2 \cdot s}$) and Bob generates ($g^{x3 \cdot x1} \parallel g^{x2 \cdot x4 \cdot s}$). Although this approximation does not capture all features of JPAKE, our required security properties are modelled, as the key establishment secrecy comes from the discrete logarithm problem and Decision Square Diffie-Hellman (DSDH) assumption over $g^{x1 \cdot x3}$ and $g^{x2 \cdot x4}$. It is of no consequence if these internal parameters are used in other protocol blocks, either. We define lemma key_establishment to verify that there is a session key establishment between Alice and Bob. Our approximation model is presented in Fig. 7 and can be used in the analysis of protocol suites.

rule **Register_JPAKE**:
1.[Fr(~pass),Fr(~nr)]-->
2.[!Alice($A,~pass,~nr ^~pass),!Bob($B,~pass, ~nr ^~pass)]

rule **Reveal_password**:
1.[!Alice(A,key,pass)]--[Reveal(A)]->[Out(pass)]

rule **Role_A**:
1.[Fr(~secA), !Alice(A, pass, keyA)]--
2.[Send(A,~secA), Key_A(A,keyA), Secret(~secA)]->
3.[Out(<senc(~secA,keyA)>), !JPAKE(A,keyA)]

rule **Role_B**:
1.let secB = sdec(msg,keyB) in
2.[In(<A,msg>), !Bob(B, pass, keyB)]--
3.[Key_B(B,keyB), Recv(B,secB), Secret(secB)]->
4.[ST_B(B,pass,keyB), !JPAKE(B, keyB)]

Figure 8: JPAKE as built-in primitive.

Built-in primitive. A simple, straightforward, yet powerful approach is to provide JPAKE as a built-in primitive under *perfect cryptography* similar to PKC, signature, hash, and symmetric encryption. Abdalla et al. [19] thoroughly proved the security of JPAKE under a rigorous key establishment model. Providing JPAKE as a built-in primitive similar to PKC cannot be the ultimate solution when other blocks reuse some JPAKE parameters. However, this approach can simplify the model. Similar to other primitives, JPAKE can be as simple as invoking predefined persistent facts (!JPAKE()), the same with other predefined primitives such as !PSK(), !Ltk(), and !Pk(). We implemented a JPAKE built-in primitive as Fig. 8 so it can be readily used.

We believe our two approaches can be effective in modelling complex algorithms in practical settings.

4.2 Case Study

We present our case studies by formally analysing MQTT and CoAP. Although CoAP and MQTT share similar design principles for constrained IoT applications, their approaches to security and implementations are quite different. In terms of lightweight protocol design, CoAP's security depends purely on Datagram TLS (DTLS) [8].

MQTT. MQTT recommends the use of the latest TLS architecture with X.509 certificates. To minimize the overhead of the TLS handshake, MQTT recommends the use of TLS session resumption via either the session ID or session ticket methods. We model the TLS 1.2 session resumption protocol from MQTT's official open source release. This will be a valuable basis since there are many MQTT variations and modes such as MQTT for Sensor Networks (MQTT-SN) [16].

Tamarin verified that MQTT's TLS 1.2 session resumption implementation is solid, where the PSK/PKC resumption has no attacks. For some applications that require PFS, we also implemented a lemma that can verify PFS in the resumption process.

CoAP The fundamental design philosophy of DTLS is to 'reconstruct' TLS over datagram packets as closely as possible, minimizing new security inventions and maximizing the amount of TLS infrastructure reuse. DTLS addresses two problems that normal TLS would have due to the unreliability of datagram transport. First, in normal TLS with stream ciphers, records depend on each other, and hence decryption of individual records is not possible. Second, TLS uses implicit sequence numbers to protect against replays and reordering. DTLS solves this by avoiding stream ciphers completely and by using explicit sequence numbers. It also employs a stateless cookie technique to protect against DoS attacks. As cryptography is perfect in the modeller, modelling DTLS thus has to take only two aspects into account, compared to TLS 1.2: 1) a MAC in each record with an explicit sequence number, and 2) the stateless cookieour . Our DTLS model can serve as a base to model other IoT protocols that are based on this protocol. We also implemented key security verification under D-Y, eCK, and PFS.

Table 2 shows our case studies including SigFox, LoRa, JPAKE, MQTT, and CoAP. Based on our initial implementation, service providers and vendors can further extend the analysis according to their application settings. We plan to continue formal analysis of other IoT protocols and will release our implementations.

5 DOS ATTACK VULNERABILITY OF IOT PROTOCOLS

IoT applications face emerging adversaries working for different motives. Thus, introducing new attack models in the formal analysis is essential for practical IoT applications. In this section, we used DoS attacks as an example of introducing emerging attack models in the IoT. Vulnerability to DoS attacks is a classic problem. We present our DoS attack modelling using our approach to formal security analysis. This will be a base for introducing application-specific adversary models.

We usually assume the adversary's goals to be to disrupt, subvert, or destroy a network, resulting in diminished or eliminated capability. A DoS attack does not require as much effort as other cryptographic attacks, but it works very well in the IoT [45]. DoS attacks can occur on the physical layer all the way to routing and application layer. However, we only consider cryptographic DoS attacks in IoT applications here. Cryptographic DoS attacks are extremely effective in terms of depleting/exhausting constrained resources such as computation, communication, and energy of battery-powered devices [47]. The crucial advantage cryptographic DoS attacks enjoy is that they cause intensive computation.

DoS attacks targeting IoT devices. To show the impact of cryptographic DoS attacks, we performed extensive experiments on a constrained IoT device, Openmote [9]. Openmote represents our battery-running constrained IoT device, as it features Cortex-M3 processors with up to 32 Mhz clock, 256/512 kB ROM, 16/32 kB RAM, and tamper-resistant key storage. Even lightweight Elliptic Curve Digital Signature Algorithm (ECDSA) signature verification/generation requires 2.6 and 4.1 seconds, respectively, on Openmote (see Table 3).

Operation	Time	Energy
Multiplication	269 ms, 24.3 mA	13.7 mJ
Exponentiation	695 ms, 24.3 mA	35.4 mA
Pairing	1,964 ms, 25.1 mA	103.5 mJ
ECDSA generation	4,104 ms, 23.6 mA	203.3 mJ
ECDSA verification	2,631 ms, 23.6 mA	130.3 mJ
AES-256 (200 byte)	1.8 ms, 20.8 mA	78.6 μJ
SHA-256 (200 byte)	3.1 ms, 20.8 mA	135.4 μJ
Radio (127 byte)	1,301 ms, 24.8 mA	67.7 mJ

Table 3: Computation overhead summary on Openmote using NIST P-256 elliptic curve. RSA will cause significantly higher overheads.

In such an environment, simply flooding the network with fake signatures or session re-initialization requests can accomplish the goal of a DoS on IoT devices. DoS attacks on IoT servers aim for amplification by transmitting handshake (re)initialization and service requests.

DoS attacks on JPAKE. Although the security of JPAKE was initially proved by Hao et al. [29] and later thoroughly by Abdalla et al. [19], they both emphasized that DoS attacks are a rare but a powerful attack on JPAKE. This is due to the intrinsic nature of PAKE protocols, where the password is the only secret between two parties and it is a low-entropy secret. Thus, there is practically no way to verify the legitimacy of a party until the key between the two parties is established.

Since the original JPAKE implementation is too heavy for constrained IoT devices, THREAD proposes to use an elliptic curve variant of JPAKE (EC-JPAKE) [2], using the NIST P-256 elliptic curve. EC-JPAKE is a more suitable choice for IoT applications since Elliptic Curve Cryptography (ECC) requires significantly shorter keys than RSA for the same level of security (e.g., 256 bit ECC $\equiv$ 3072 bit RSA), resulting in lower performance and memory requirements for constrained IoT devices. ECC-based security schemes have been a favourable choice for IoT and WSNs applications, as they can provide higher security even on constrained devices.

Although the original JPAKE consists of 4 flights in total, EC-JPAKE contracts it to 3 flights for communication efficiency (see Fig. 9). Each EC-JPAKE party must perform 14 exponentiations and 8 multiplications to derive the shared key with 3 protocol flights. We measured the computation/communication cost of the EC-JPAKE handshake on Openmote using the 802.15.4 radio. One EC-JPAKE

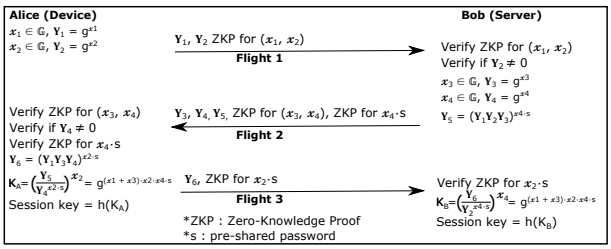

Figure 9: EC-JPAKE protocol overview.

Operation	Computation	Time	Energy
Flight 1	$4ex + 2m$	3.3 s	169.3 mJ
Flight 2	$8ex + 6m$	7.1 s	366.0 mJ
Flight 3	$2ex$	1.3 s	70.9 mJ
Total computation 14 ex + 8 m		11.8 s	**606.2** mJ
Including radio activities		20.5 s	**1195.2** mJ

Table 4: Average EC-JPAKE overhead of Alice on Openmote. Each EC-JPAKE session requires 20.5 seconds and consumes 1195.2 mJ (0.14 % of a CR-2032 battery capacity).

handshake consumes 11.8 seconds for computation and 20.5 seconds with communication included. One handshake consumes 1195.2 mJ, which is 0.14% of a new CR-2032 battery capacity (see Table 4). Exploiting the fault-tolerant nature of IoT devices, continuous false authentication requests can deplete the device energy or can interfere with legitimate operations.

Modelling DoS attacks. We now show how to model such DoS attacks using the Tamarin prover. To the best of our knowledge, this is the first modelling of DoS attacks in the IoT using a formal security verification tool. We reuse the SigFox PKC example here for its simplicity and because all SigFox packets carry a signature. In rule Receive, we defined a state fact ST_B_1(B, ltkB, pkA, A,na) that can be used in later steps. Let us assume Bob's return message is cryptographically heavy (an ECDSA signature generation itself is indeed heavy on Openmote. 4.1 seconds). If an adversary can

rule **Send_back**:
1.[St_B_1(B,ltkB, pkA, A,na, pkB),Fr(~nb)]--
2.[B_Send(B,aenc(<A, na>, pkB)),
DoS_Protection(aenc(<A, na>, pkB))]
3.->[Out(sign(<B,~nb>,ltkB))]

lemma **DoS_protection**:
"All m #i. DoS_Protection(m) @i ==> (Ex A #j. Send(A,m) @j & Honest(A)@j) | (Ex A #r. Reveal(A)@r & Honest(A)@i& r<i)"

Figure 10: Our DoS attack verification implementation.

Protocol	Result
JPAKE	Attack on key establishment
JPAKE with our SP	Proof
CoAP-PSK	Proof
CoAP-PKC	Attack on signature
MQTT	Attack on signature
SigFox	Attack on signature
LoRa	Proof

Table 5: Overview of DoS attack verifications.

Approach	Lightweight	Hardness control	Communication
One-way hash [44]	O	X	O
Cookie [1]	O	X	X
Time lock [40]	X	O	O
Client puzzle [31]	O	O	X
Our approach	O	O	O

Table 6: Comparison of DoS attack countermeasures.

force this step an unlimited number of times, this is a DoS attack vulnerability. We define lemma DoS_protection to verify whether there are B_Send actions without a legitimate sender with verified signature. This lemma will be falsified if any contradictory case is found. The lemma DoS_protection guarantees that there is no such exhaustion in the rule Send_back. However, if we extend this to the rule Receive, where signature verification is performed, this lemma will find DoS attacks.

The same approach can be applied to the other aforementioned protocols. We summarize the result of DoS vulnerabilities in Table 5.

6 OUR DOS COUNTERMEASURE

In this section, we propose our countermeasure against DoS attacks for EC-JPAKE. Although EC-JPAKE is used as an example, the countermeasure can be used generally in other protocols since it does not rely on any internal parameters of EC-JPAKE and uses standard AES/HMAC.

The intrinsic DoS weakness of EC-JPAKE comes from the juggling and ZKP to hide the low-entropy shared password. Neither party can verify the legitimacy of the other party without key establishment. EC-JPAKE is resistant to off-line dictionary attacks since $g^{(x_1+x_3+x_4)}$ and $(x_2 \cdot s)$ are random elements in group G (under the decisional Diffie-Hellman assumption). Adversaries cannot distinguish between $(x_1 + x_2 + x_3)$ or $(x_1 + x_3 + x_4)$ under the discrete logarithm assumption. The main challenge arises here since the protocol must not leak any information on the password before the key verification. To rephrase, we cannot leverage the shared password for DoS attack protection, otherwise we will compromise the principle of the JPAKE design.

Challenge of existing countermeasures. As mentioned in Section 2, existing countermeasures pose drawbacks in the IoT. The one-way hash function is lightweight, but no longer secure in the event of a node compromise [44]; the cookie approach is only secure against attacks from spoofed

IP addresses [1]. Time-lock puzzles can adaptively control the hardness according to the adversary capability, but heavy puzzle generation can be a pitfall on constrained IoT devices [40]. As observed in Table 3, radio activities consume as much energy as cryptographic computations. Since computation and communication are at a premium in the majority of IoT applications, causing excessive packets by client puzzles can be a pitfall too. In particular, in low-power multi-hop networks such as 802.15.4 [17], where the MTU is 127B, packet fragmentation can cause more issues. To avoid this, much effort has been invested in outsourcing puzzle generation [46], but it remains a challenge in practice. As shown in Table 6, the majority of IoT protocols are vulnerable to DoS attacks.

6.1 Our Approach

One benefit of our DoS modelling is that it can detect exactly which step of the protocol is vulnerable. Our proposed countermeasure is lightweight for issuing and verifcation, and the issuer (device) can set the exact hardness adaptively according to the adversary capability. We propose to use an AES-brute-force-based server puzzle (SP) that can dynamically adjust the hardness in case of consecutive DoS attack attempts and advanced adversaries. We further use the cookie approach as an option to defend against DoS attack based on IP spoofing. Our server puzzle is similar to the traditional hash-inversion-based client puzzle, but we base it on AES encryption.

Puzzle construction. Our construction is similar to Propagating Cipher Block Chaining (PCBC), but we use multiple keys to generate the next plain text. At a high level, SP consists mainly of k-bits AES brute force problems, where k-bits control the base hardness. Similar to a client puzzle's sub-puzzle structure, we use i-round block-chain-style sub-puzzles for accurate hardness control. This is due to the exponential hardness increment of the brute force problem. For instance, an 35-bit AES-128 brute force problem required an average 117 minutes to solve, while 36-bit required an average of 940 minutes (measured on a moderately strong laptop with Intel i-5, 4-core, 3.2 GHz clock, 4GB memory, and no background processes, NIST Known Answer Test (KAT) vectors [18]).

The first puzzle starts with i=0 and increases the number of rounds or k bits upon consecutive requests or advanced adversaries. The first plain text (P_0) is generated by concatenating the request (R), the current time (T), and a random vector ($RV \in \{0,1\}^*$) as padding. Here, we use the notation $(0 \rightarrow \lambda)$ as superscript (e.g., $K_0^{(0\rightarrow\lambda)}$) to show the length of the element, which is $(\lambda - 0)$-bits long in this case.

$$K_0^{(0\rightarrow\lambda)} \in \{0,1\}^* \tag{1}$$

$$P_0 = R \parallel T \parallel RV \tag{2}$$

$$C_0 = E_{K_0}(P_0) \tag{3}$$

After picking an ephemeral AES key (K_0) of size λ-bits, the first ciphertext (C_0) is generated by using the AES encryption function (E_{key}) as $C_0 = E_{K_0}(P_0)$. The next block keys K_i, for 1,..,i are generated by using a k-bit random vector and $(\lambda - k)$-bits of the prior ciphertext.

$$K_i^{(0\rightarrow\lambda)} = RV^{(0\rightarrow k)} \in \{0,1\}^* \parallel C_{i-1}^{(k\rightarrow\lambda)} \tag{4}$$

The next plain texts P_i, for 1,..,n are XORed with the $(i-1)$-th AES key K_{i-1} to eliminate the identical ciphertext blocks and to force sequential i-round operation.

$$P_i = K_{i-1} \oplus P_{(i-1)}, \text{for i=1,..,n} \tag{5}$$

The corresponding ciphertexts ($C_{(0..i)}$) are generated using symmetric AES encryption $E_{K_i}(P_i)$.

$$C_i = E_{K_i}(P_i), \text{for i=1,..,n} \tag{6}$$

We use the HMAC function to generate short ciphertext blocks (CT_i) since the corresponding ciphertext blocks are of the same length as plaintext blocks.

$$CT_i = HMAC_{K_i}(C_i), \text{for i=0,..,n} \tag{7}$$

Our server puzzle (SP) consists of initialization vectors, i ciphertexts, and a $(\lambda - k)$-bits key. The puzzle SP is sent to the server (adversary) upon detection of any sign of DoS attacks. Our server puzzle generation is described in Fig. 11.

$$SP = T \parallel RV \parallel CT_{(0..i)} \parallel K_0^{(k\rightarrow\lambda)} \tag{8}$$

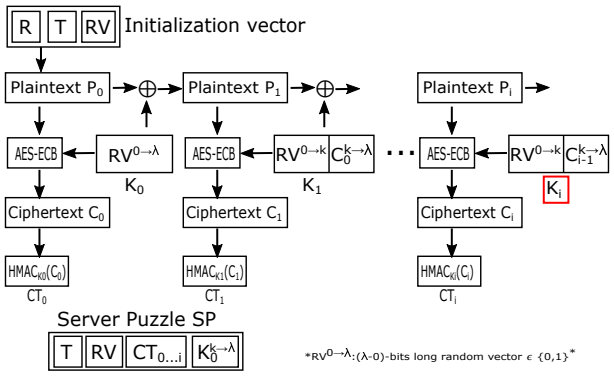

Figure 11: Our proposed server puzzle construction.

Upon receiving the server puzzle SP, the server (adversary) is required to solve a k-bit, i-round AES brute force problem (using the 'brute force function' BF).

$$K_i = BF(P_i, C_i) \tag{9}$$

The answer (SA) must contain the existing parameters with a HMAC using K_i (see Fig. 12).

$$SA = R \parallel T \parallel RV \parallel HMAC_{Ki}(R \parallel T \parallel RV) \tag{10}$$

To verify the answer SA, the device only needs to perform one HMAC operation.

Benefits. Our server puzzle construction offers four benefits for IoT applications:

(1) It can be used generally in IoT protocols since it is based on standard AES/HMAC and does not depend on other parameters.

(2) Puzzle generation requires i-round AES/HMAC symmetric operations, which are lightweight even on extremely constrained devices.
(3) Although i ephemeral AES keys are used, the device needs to store only one AES key (K_i) to verify the puzzle.
(4) The hardness of the problem is dynamically adjustable in two ways: 1) the length of the AES key k and 2) the number of rounds i. The device can control the expected time spent by the attacker by adding more rounds.

EC-JPAKE with DoS countermeasure. With our proposed server puzzle scheme and cookie approach, we add optional DoS defensive steps to EC-JPAKE (see Fig. 13). Note that these steps are optional; they are not used when there is no evidence of DoS attack in the protocol execution. We further modeled EC-JPAKE with our SP using Tamarin for DoS attack verification. Tamarin verified EC-JPAKE with our SP as resistant to DoS attacks.

7 CONCLUSION

In this paper, we presented a practical automated formal analysis of IoT protocols using the Tamarin prover under D-Y, eCK, and PFS. Upon protocol model/code changes, formal symbolic models will require minimal effort to perform substantial full analysis. We investigated the critical challenges of formal security analysis tools in the IoT and proposed two solutions, which will be useful in the analysis of complicated protocol scenarios.

Furthermore, we showed how to represent DoS attacks in the model; we showed that the majority of IoT protocols are vulnerable to cryptographic DoS attacks. To protect IoT devices from such attacks, we proposed a server puzzle that can be used generally in any IoT protocols. Our server puzzle features lower computation and communication complexity for use in constrained IoT devices, yet offers control over hardness.

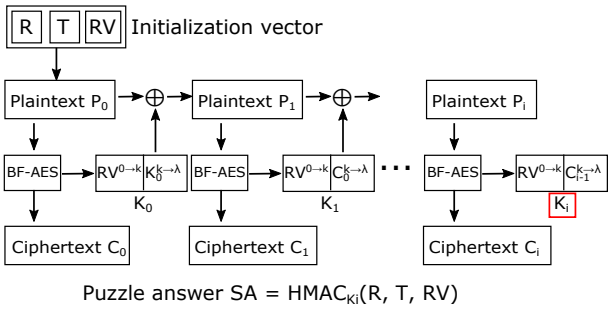

Figure 12: Server puzzle solving procedure.

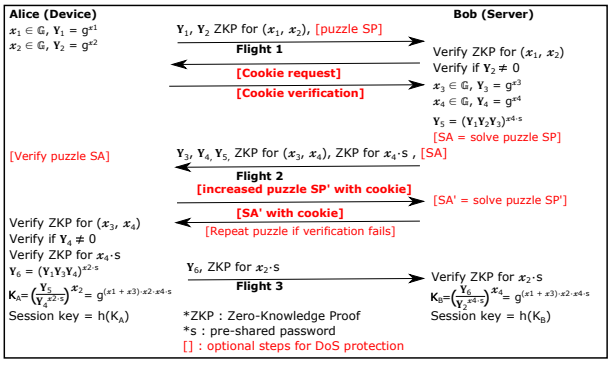

Figure 13: EC-JPAKE protocol with our proposed DoS attack countermeasure.

ACKNOWLEDGMENTS

The authors would like to thank Prof. Cas Cremers and his students Katriel Cohn-Gordon and Dennis Jackson for their theoretical and practical Tamarin modelling support. The authors would like to express gratitude to the UNSW Innovation connection project and WBS Technology staff, Walter Huang, Jimmy Chan and Brian Cheney for their practical support. The authors would also like to thank the anonymous referees for their valuable comments and helpful suggestions. This research is supported in part by the Australian Research Council project DP150100564.

REFERENCES

[1] 2010. https://tools.ietf.org/html/rfc5996. (2010). [Online; accessed 24-May-2017].
[2] 2010. https://tools.ietf.org/html/draft-cragie-tls-ecjpake-00. (2010). [Online; accessed 24-May-2017].
[3] 2013. The DDoS That Almost Broke the Internet. https://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet/. (2013). [Online; accessed 24-May-2017].
[4] 2014. https://tools.ietf.org/html/rfc7252. (2014). [Online; accessed 24-May-2017].
[5] 2015. http://www.symantec.com/connect/blogs/iot-worm-used-mine-cryptocurrency. (2015). [Online; accessed 24-May-2017].
[6] 2015. http://tools.ietf.org/html/draft-hao-jpake-01/. (2015). [Online; accessed 24-May-2017].
[7] 2015. http://threadgroup.org/RESOURCES/White-Papers. (2015). [Online; accessed 24-May-2017].
[8] 2015. https://tools.ietf.org/html/rfc6347. (2015). [Online; accessed 24-May-2017].
[9] 2015. http://www.openmote.com/. (2015). [Online; accessed 24-May-2017].
[10] 2016. http://www.emarketer.com/Article/Security-Top-Barrier-Internet-of-Things-Growth/1013624. (2016). [Online; accessed 24-May-2017].
[11] 2016. https://github.com/tamarin-prover/tamarin-prover. (2016). [Online; accessed 24-May-2017].
[12] 2016. 2016 Dyn cyberattack. https://techcrunch.com/2016/10/21/many-sites-including-twitter-and-spotify-suffering-outage/. (2016). [Online; accessed 24-May-2017].
[13] 2017. https://www.lora-alliance.org/What-Is-LoRa/LoRaWAN-White-Papers. (2017). [Online; accessed 24-May-2017].
[14] 2017. https://www.sigfox.com/. (2017). [Online; accessed 24-May-2017].
[15] 2017. http://mqtt.org/. (2017). [Online; accessed 24-May-2017].
[16] 2017. http://mqtt.org/2013/12/mqtt-for-sensor-networks-mqtt-sn. (2017). [Online; accessed 24-May-2017].
[17] 2017. http://www.ieee802.org/15/pub/TG4.html. (2017). [Online; accessed 24-May-2017].
[18] 2017. http://csrc.nist.gov/groups/STM/cavp/documents/aes/KAT_AES.zip. (2017). [Online; accessed 24-May-2017].

[19] Michel Abdalla, Fabrice Benhamouda, and Philip MacKenzie. 2015. Security of the J-PAKE password-authenticated key exchange protocol. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 571–587.

[20] Nadhem J Al Fardan and Kenneth G Paterson. 2013. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 526–540.

[21] Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, P Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, et al. 2005. The AVISPA tool for the automated validation of internet security protocols and applications. In *International Conference on Computer Aided Verification*. Springer, 281–285.

[22] 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.

[23] David Basin, Saša Radomirovic, and Lara Schmid. 2016. Modeling Human Errors in Security Protocols. (2016).

[24] Steven M Bellovin and Michael Merritt. 1993. Cryptographic protocol for secure communications. (Aug. 31 1993). US Patent 5,241,599.

[25] Bruno Blanchet, Martín Abadi, and Cédric Fournet. 2008. Automated verification of selected equivalences for security protocols. *The Journal of Logic and Algebraic Programming* 75, 1 (2008), 3–51.

[26] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. 2016. Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE.

[27] Thai Duong and Juliano Rizzo. 2011. Here come the XOR Ninjas. *White paper, Netifera* (2011).

[28] Dov M Gabbay, Christopher John Hogger, and John Alan Robinson. 1998. *Handbook of Logic in Artificial Intelligence and Logic Programming: Volume 5: Logic Programming*. Clarendon Press.

[29] Feng Hao and Peter Ryan. 2010. J-PAKE: authenticated key exchange without PKI. In *Transactions on computational science XI*. Springer, 192–206.

[30] David P Jablon. 2001. Cryptographic methods for remote authentication. (May 1 2001). US Patent 6,226,383.

[31] Ari Juels and John G Brainard. 1999. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks.. In *NDSS*, Vol. 99. 151–165.

[32] Deepak Kapur, Paliath Narendran, and Lida Wang. 2003. An E-unification algorithm for analyzing protocols that use modular exponentiation. In *International Conference on Rewriting Techniques and Applications*. Springer, 165–179.

[33] Jun Young Kim, Wen Hu, Dilip Sarkar, and Sanjay Jha. 2017. ESIoT: enabling secure management of the internet of things. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 219–229.

[34] Jun Young Kim, Wen Hu, Hossein Shafagh, and Sanjay Jha. 2016. SEDA: Secure Over-The-Air Code Dissemination Protocol for the Internet of Things. *IEEE Transactions on Dependable and Secure Computing* (2016).

[35] Ralf Küsters and Tomasz Truderung. 2009. Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In *Computer Security Foundations Symposium, 2009. CSF'09. 22nd IEEE*. IEEE, 157–171.

[36] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. 2007. Stronger security of authenticated key exchange. In *International Conference on Provable Security*. Springer, 1–16.

[37] Christopher Lynch and Catherine Meadows. 2004. Sound approximations to Diffie-Hellman using rewrite rules. In *International Conference on Information and Communications Security*. Springer, 262–277.

[38] Sebastian Mödersheim. 2011. Diffie-Hellman without difficulty. In *International Workshop on Formal Aspects in Security and Trust*. Springer, 214–229.

[39] Long Ngo, Colin Boyd, et al. 2011. Automated proofs for Diffie-Hellman-based key exchanges. In *Computer Security Foundations Symposium (CSF), 2011 IEEE 24th*. IEEE, 51–65.

[40] Ronald L Rivest, Adi Shamir, and David A Wagner. 1996. Time-lock puzzles and timed-release crypto. (1996).

[41] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. 2012. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *2012 IEEE 25th Computer Security Foundations Symposium*. IEEE, 78–94.

[42] Benedikt Schmidt, Ralf Sasse, Cas Cremers, and David Basin. 2014. Automated verification of group key agreement protocols. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 179–194.

[43] Claus-Peter Schnorr. 1991. Efficient signature generation by smart cards. *Journal of cryptology* 4, 3 (1991), 161–174.

[44] Hailun Tan, John Zic, Sanjay K Jha, and Diethelm Ostry. 2011. Secure multihop network programming with multiple one-way key chains. *Mobile Computing, IEEE Transactions on* 10, 1 (2011), 16–31.

[45] Eugene Y Vasserman and Nicholas Hopper. 2013. Vampire attacks: draining life from wireless ad hoc sensor networks. *IEEE transactions on mobile computing* 12, 2 (2013), 318–332.

[46] Brent Waters, Ari Juels, J Alex Halderman, and Edward W Felten. 2004. New client puzzle outsourcing techniques for DoS resistance. In *Proceedings of the 11th ACM conference on Computer and communications security*. ACM, 246–256.

[47] Anthony D Wood and John A Stankovic. 2002. Denial of service in sensor networks. *computer* 35, 10 (2002), 54–62.