SEMUD: Secure Multi-hop Device-to-Device Communication for 5G Public Safety Networks

Milan Schmittner, Arash Asadi, and Matthias Hollick
Secure Mobile Networking Lab, Technische Universität Darmstadt, Germany
{milan.schmittner, arash.asadi, matthias.hollick}@seemoo.tu-darmstadt.de

***Abstract*—Multi-hop Device-to-Device (D2D) communication has emerged as a key enabler for public safety applications in 5G networks. A failure of these applications would be catastrophic since they control vital human-in-the-loop services. As a result, it is of utmost importance to identify and mitigate security risks prior to commercial deployment. To this end, we devise SEMUD, the first secure multi-hop D2D solution for 5G mobile networks. It enables robust and secure communication even in the absence of supporting infrastructure. We design SEMUD to be compliant with 3GPP Proximity-based Services, the standard to implement D2D communications. We implement SEMUD in the ns-3 simulator and our testbed. Via extensive simulation, we assert its resilience against strong adversaries and attacks. Furthermore, we show the energy efficiency and achievable throughput of our proposal on real devices.**

I. Introduction

Although D2D was initially designed for single-hop communications, it re-emerged in 5G cellular networks for multi-hop communications that are indispensable for the new generation of national security and public safety communication systems [1]–[4]. However, due to the initial single-hop design of D2D communication, its multi-hop challenges, i.e., interference and resource management and security, are under-explored. In particular, security is ignored as many assume D2D communication to be inherently secure due to the presence of cellular infrastructure. This is indeed a dangerous misconception as we have observed a plethora of security issues in the infrastructure throughout the evolution cellular networks [5]. These issues illustrate that design trade-offs made a decade ago are no longer effective in 5G.

Moreover, the infrastructure's role in D2D communications reduces from a managing entity to a facilitator. Hence, D2D users are exposed to even more threats in comparison to legacy cellular users. These threats are particularly hard to circumvent in multi-hop D2D because the presence of intermediary users subjects the system to several security threats such as blackhole and wormhole attacks as well as selfish nodes. It is of utmost importance to protect multi-hop D2D from these attacks due to its critical use cases including public safety communication. Prior work in secure D2D communications focused on authentication and key management [6]. However, the 3GPP Proximity-based Service (ProSe) architecture facilitates these issues by amending the legacy system design with D2D specific network entities [7], [8]. Nevertheless, end-to-end security is still a challenge for multi-hop D2D [4], as current solutions do not protect users from the above advanced attacks.

ISBN 978-3-901882-94-4

In this paper, we propose a framework for Secure Multi-hop D2D (*SEMUD*) communication. SEMUD is the first framework to address security issues for multi-hop D2D and to provide a standard-compliant solution resilient against a wide spectrum of attacks in presence and absence of cellular infrastructure. We design SEMUD to be compatible with the 3GPP ProSe architecture, which is specifically conceived to cater the needs of D2D applications such as certificate management and neighbor discovery through the ProSe architecture. At its core, SEMUD's communication protocol provides authenticated, confidential, and denial-of-service (DoS)-resilient end-to-end message exchange by leveraging state-of-the-art security mechanisms such as Merkle tree message labeling and flow-based neighbor reliability metrics [9], [10]. The main contributions of this paper are:

- Compared to its competitors, SEMUD incurs low overhead w.r.t. computation and communication while achieving stronger security properties by implementing opportunistic flow initialization, efficient duplicate detection, and an in-network Merkle tree compression algorithm.
- We compare the performance of state-of-the-art cryptographic primitives and implement SEMUD both in real hardware and the ns-3 simulator. We benchmark SEMUD against a popular secure multi-hop scheme to assess its performance against two strong adversaries (blackhole, and combined replay and blackhole attack) using simulation; and we demonstrate the practicality of SEMUD assessing its power consumption and computational overhead on representative hardware platforms.

The rest of this paper is structured as follows: in Section II, we give an overview of SEMUD's architecture and workflow, while we describe SEMUD communication in Section III. In Section IV, we shed light on our implementation on which we base our evaluation following in Section V. We discuss related work in Section VI and finally conclude in Section VII.

II. Secure Multihop D2D (SEMUD)

In this section, we state our security assumptions and provide an overview of our 3GPP ProSe-compliant architecture as well as SEMUD's workflow.

A. Security Assumptions

Our security assumptions consist of an attacker model and a trust model. We elaborate on them in the following.

Attacker Model. In this work, we consider attacks on the classic security triad Confidentiality, Integrity, and Availability (CIA). In particular, our attacker is an entity controlling a portion of authenticated User Equipments (UEs) within the network. She can consequently take part in normal network operations, but is not limited to, mounting localized jamming, message injection, modification, and dropping attacks; specific attacks on the forwarding protocol; or any combination thereof. However, the attacker cannot break cryptographic primitives. The attacker may choose time and location of the attack. This means, she might start attacks when the infrastructure network is (temporarily) unavailable and, e. g., cannot revoke certificates from misbehaving nodes or otherwise help in mitigating attacks.

Trust Model. SEMUD devises an end-to-end communication scheme. Hence source and destination UEs need to trust each other. This trust establishment is mediated by the trusted infrastructure. However, we do not assume trust relationships between source/destination and other intermediate UEs since this would either: (i) impose a scalability problem with the complexity of $O(n^2)$; or (ii) require continuous access to the infrastructure network which cannot be guaranteed.

B. SEMUD Architecture

To foster D2D services, 3GPP includes three ProSe-specific elements in the existing network architecture. These elements are ProSe Application, ProSe Function, and ProSe Application Server. Fig. 1 graphically depicts the location and interworking of each element in 3GPP's architecture. In addition, the other existing network entities such as Mobility Management Entity (MME) and Home Subscriber Server (HSS) are amended to support communication with these new ProSe-specific elements. Here, we briefly describe the main ProSe elements and refer the interested reader to [7] for more details.

ProSe Application. The ProSe Application runs on the ProSe-enabled UEs and it supports control and data communication between ProSe-enabled UEs and the ProSe Function. Moreover, direct discovery procedure is handled by ProSe Applications. SEMUD ProSe Application supports direct neighbor discovery, network authentication, and message forwarding. These operations are explained in detail in the following subsections. SEMUD ProSe Application also: (i) has access to a pseudo-random number generator $rand\,()$, (ii) can compute a cryptographic hash function $hash\,(\cdot)$, (iii) has access to a stream cipher $prf\,(K, n)$ which takes a key K and some nonce n as inputs, and (iv) can compute authentication tags $tag\,(K, \cdot)$ based on a shared key K. We discuss suitable candidate functions in Section IV-B.

ProSe Function. The ProSe Function is a logical function located inside Evolved Packet Core (EPC), and it handles network related actions of ProSe. In the current specification [7], ProSe Function may play different roles depending on the use-case. SEMUD ProSe Function mainly assists in direct discovery, EPC-level discovery, and direct communication. The location information used for EPC-level discovery is obtained from the SUPL Location Platform (SLP). ProSe function also acts as a reference point between SEMUD UEs and SEMUD Application Server that is used for operations such as certificate distribution.

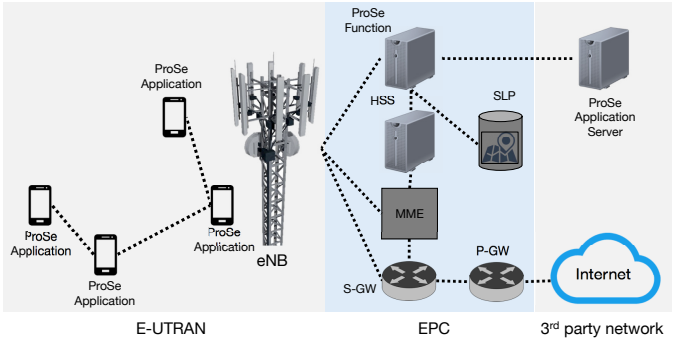

Fig. 1: 3GPP ProSe reference architecture.

ProSe Application Server. This entity is located outside the EPC and it provides the necessary functionalities for the SEMUD specific operations. It also supports application layer functionalities such as the storage of ProSe User and ProSe Function IDs, and mapping between them. The SEMUD Application Server acts as a Certificate Authority (CA) that is accessible during normal network operation. During this time UEs can contact the CA to receive a valid certificate. These certificates are used for mutual user authentication. This means that registering new users is only possible while the infrastructure network is available. However, secure communication is possible between registered users even when the Application Server is inaccessible.

C. SEMUD Workflow

We design SEMUD as a comprehensive end-to-end solution. Unlike existing secure multi-hop frameworks [9]–[11], SEMUD explicitly leverages cellular infrastructure as a trust anchor while remaining operational in its absence. In the following, we elaborate on the workflow of SEMUD.

Phase 1: Subscription. As for other ProSe services, a user interested in SEMUD should send a subscription request including its public key to the SEMUD ProSe Function. The ProSe Function forwards this request to the ProSe Application Server and requests for authorization from HSS to use ProSe services for this user. If both requests are accepted, ProSe Function forwards the successful subscription message to the UE. This message contains a certificate for the UE's public key.

Phase 2: Discovery and Authentication. ProSe offer two device discovery methods, namely, EPC-level discovery and direct discovery. SEMUD can benefit from both methods. UEs associated with SEMUD services can use the infrastructure (i.e., EPC-level discovery) to find and join other SEMUD associated UEs in vicinity. In EPC-level discovery, neighbors' public keys and certificates are provided by the network. In the absence of infrastructure, SEMUD UEs exploit direct

discovery and use beacons to explore their vicinity. A UE can transmit beacons either to advertise its presences or to request a response from others UEs. While EPC-level discovery is more efficient, direct discovery is crucial when infrastructure support is nonexistent. In direct discovery, neighbors' public keys and certificates are exchanged during the association procedure [7]. Neighbor (i.e., UEs in a single-hop distance) authentication prevents Sybil and blackmailing attacks on the forwarding protocol described below. A symmetric session key between two neighbors can be derived using a Diffie–Hellman key exchange. Neighbor-to-neighbor communication can then be authenticated.

Phase 3: Key Distribution. SEMUD uses public keys for user authentication and session key establishment. In the presence of infrastructure, key distribution is done by querying the ProSe Application Server. In the absence of infrastructure, a UE requests a destination's public key and a certificate in the association stage. Thus, the destination is authenticated in the same way as above and a shared secret derived using Diffie–Hellman key exchange. We denote such a shared key between source s and destination d as K_{sd}.

Phase 4: Communication. This phase is rather simple in legacy single-hop D2D systems and consists of direct communication between the D2D pairs over the pre-allocated/negotiated frequency band [12]. However, multi-hop D2D communication is exposed to new security risks that are not present in legacy single-hop D2D systems. Hence, SEMUD communication phase requires a comprehensive procedure from message generation to acknowledgment handling to ensure its robustness against a wide spectrum of attacks such as message forgery, blackhole, and wormhole attacks.

Phase 5: Termination. The termination process is as defined in [7] which consist of sending a deregistration message to the ProSe Function. Next, the user certificate is added to the ProSe Application Server certificate revocation list.

III. Communication Phase in SEMUD

In this section, we describe how SEMUD secures multi-hop D2D communication in a computational and bandwidth efficient manner. Any effective secure communication mechanism should follow the CIA triad model. Thus, SEMUD exploits symmetric key cryptography for message confidentiality and integrity and an adaptive neighbor reliability metric for availability. Next, we elaborate on the design and communication procedure of SEMUD (see Fig. 2 for an overview).

A. Message Generation

SEMUD establishes flows for end-to-end communication similar to [11]. The cryptographic material securing each flow is drawn from a Merkle tree, an accepted cryptographic tool for secure multi-hop communication [13], [14]. We first introduce the message format and then discuss peculiarities of Merkle tree usage and construction.

Message Format. The SEMUD MSG in Eq. (1) contains flow synchronization (SYN) and automatic repeat request (ARQ) *flags*, source s and destination d identifiers, flow identifier H which is the root of a Merkle tree of height l, the kth MSG identifier b_k, and an integrity check value icv. If SYN is set, a nonce n is included as well. The user payload may be encrypted using the stream cipher $prf\left(K_{sd}, hash\left(n+k\right)\right)$. The icv is computed over all fields excluding the flow authenticator f_k and length l' as well as ARQ. SYN is set until the first ACK of the flow is received. Additional meta data (message type, length of hash values, and length of entire message) is excluded for brevity.

$$MSG = \langle flags, s, d, H, b_k, icv, n, f_k(l', l), \mathcal{P} \rangle \tag{1}$$

Merkle Tree Usage. SEMUD utilizes Merkle hash trees for message labeling, flow authentication, and proof of message reception. In particular, the idea is to use the input values for the tree's leaf nodes as message identifiers b_k and to commit to them with the root H which is used as a flow identifier. The messages identifiers, in turn, are computed from a secret a_k as $b_k = hash\left(a_k\right)$. Since MSGs are end-to-end authenticated, the destination node will only reveal the pre-image a_k of the message identifier b_k for authentic messages in the form of an acknowledgment (ACK) (Section III-D). Upon reception of a_k, intermediate UEs can deduce that the destination must have received an authentic message with b_k.

SEMUD Merkle Tree Construction. We construct the SEMUD Merkle tree as follows: (i) we use a unique and random nonce n and use it together with K_{sd} to seed a cryptographically secure pseudo-random number generator $prf\left(\cdot,\cdot\right)$, e. g., a stream cipher; and (ii) we "chop" the output of $prf\left(K_{sd}, n\right)$ into w blocks of size $\left|hash\left(\cdot\right)\right|$ to create all a_k for $k = 1, \ldots, w$ and construct the Merkle tree as shown in Fig. 3. Our unique approach of seeding the Merkle tree with a nonce n enables the source to share all secret values a_k with the destination by just communicating n. Together with the shared key K_{sd}, the destination is able to repeat the Merkle tree construction process and retrieve all a_k. Without (i), the source should communicate all a_k individually in a confidential manner that would waste bandwidth.

When creating the Merkle tree from n and K_{sd}, we need to assert that n is never reused for any source–destination pair, otherwise replay attacks are possible. Reasonable candidates for n are timestamps or randomly chosen values drawn, e. g., from a system-provided $rand\left()$ function. The drawback of choosing timestamps as nonces is the additional attack vector on time synchronization services, e. g., NTP [15] or GPS [16]. When choosing n purely at random, n must be large enough to avoid nonce reuse due to the well-known "birthday problem". We choose to implement the second option with a random 192-bit nonce.

The tree size w is optimally chosen such that it is equal to the number of messages a source node wishes to transmit for a certain flow. If this number is known a priori, w can be approximated and fed into SEMUD as an optimization. In all other cases, SEMUD has to rely on a default tree size. However, choosing the default tree size incurs a trade-off: (i) the length of the flow authenticator included in every message grows logarithmically with the tree size, but (ii) very

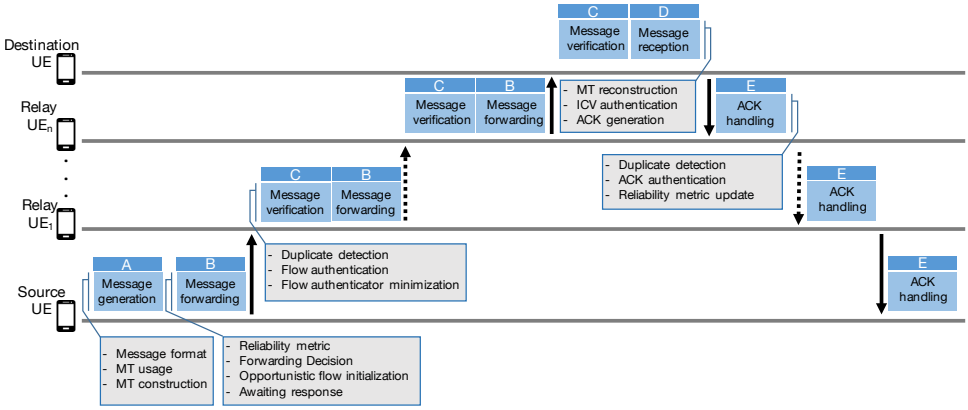

Fig. 2: Overview of SEMUD communication stages and the operations executed within each stage.

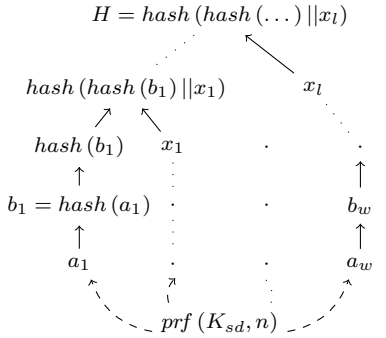

Fig. 3: SEMUD Merkle tree generation. The leaf seeds a_k are drawn from a stream cipher $prf\,(\cdot,\cdot)$, which, in turn, is seeded by a secret key K_{sd} and a public nonce n.

small trees cause frequent flow restarts, i. e., whenever all b_k have been used, a new tree must be created. In the following sections, we propose countermeasures to reduce the impact of flow restarts (Section III-B) and to dynamically minimize the flow authenticator length (Section III-C).

B. Message Forwarding

Here, we describe the forwarding decision that is based on a reliability metric. Moreover, we explain how established flows are used to initialize new flows.

Reliability Metric. In contrast to the per-destination routing state used in classic MANET protocols, SEMUD keeps the forwarding state per flow. Thus, SEMUD UEs maintain separate per-neighbor reliability metrics for every encountered flow. The reliability metric $r \in [0, 1]$ is computed as the running average of the MSG delivery rate (i. e., the number of valid ACKs returned from a neighbor) as described in [11]. It has been shown in [11], [17] that this approach provides lightweight protection against any type of accidental and deliberate message loss including hard-to-detect selective message dropping, i. e., greyhole attacks.

Forwarding Decision. This decision is made probabilistically based on the reliability metric. A UE forwards a MSG with probability $1 - r$ using a broadcast transmission or with probability r using a unicast to the most reliable neighbor. Should two or more neighbors have the same reliability metric, we use the average round-trip time to break the tie. The intuition is that we use broadcast for "route exploration" if no reliable path exists, and otherwise unicast for "route exploitation".

Opportunistic Flow Initialization. New flows in SEMUD will be established (i) when a new source–destination pair wishes to start communication, or (ii) upon *flow restarts*, i. e., when all leafs of an existing flow's Merkle tree have been used up, but the source–destination pair wants to continue communication. In a straight-forward implementation, forwarding UEs would initialize the reliability metrics for newly encountered flows with 0, which would mean that initial MSGs would always be broadcast, i. e., flooded through the entire network. However, we propose to allow the reliability metrics of a new flow to be copied from an existing flow when (i) flows restart (source and destination of old and new flow matches) or (ii) the UE already knows a path to the destination which has worked for a different flow (only the destination of old and new flows matches). If multiple candidate matches exist, we choose the most recently updated flow to assure freshness. On the one hand, copying routing state results in information reuse, i. e., unnecessary network-wide flooding due to route exploration for an already discovered destination is avoided. On the other hand, it maintains the strong security properties assured by separating routing state. Note that in case of an impersonation attack, opportunistic flow initialization might result in choosing the compromised flow. Nevertheless, such mistakes are quickly mitigated due to quick reliability updates calculated from valid ACKs or their timeouts. We will demonstrate the advantage of our opportunistic flow initialization approach in Section V.

Awaiting Response. When forwarding a MSG, a UE starts one timer for each new b_k which expires after T_{ACK}. In addition to starting the timer, the tuple $\langle m, b_k, H\rangle$ together with the forwarding decision is added to a collection of previously seen MSGs. This tuple serves the purpose of authenticating ACKs (as described in Section III-E) and it is discarded after the MSG timer expires. If the timer expires and no ACK has been received, the reliability metric for the next hop UE is decreased. We employ an adaptive TCP-inspired timeout calculation for T_{ACK} following the same approach as in [18].

C. Message Verification

Next, a UE filters out the MSGs that either have already been forwarded (i.e., duplicates) or contain an invalid flow authenticator. In addition, a UE computes the minimal authenticator length for the next hop UE.

Duplicate Detection. Duplicate detection consists of two steps. First, a UE calculates a message digest m using a collision-resistant hash function of the incoming MSG (excluding the variable-length field f_k). This serves for identifying unique MSG copies which might have the same message identifier b_k. If the UE has already seen the pair $\langle m, b_k\rangle$, the MSG is dropped.

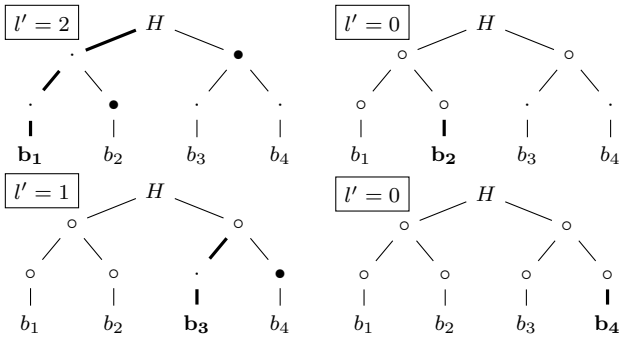

Fig. 4: Exemplary Merkle tree ($w = 4$) visualizing optimal flow authenticator lengths l' for different MSG identifiers b_k. Bullets (•) indicate tree nodes that have to be included in MSG k. Circles (○) are known tree nodes sent in previous MSGs. Dots (·) are unknown nodes but are not required to authenticate b_k. Thick lines indicate the verification path.

We propose that each UE keeps per-flow state to memorize which MSGs have already been acknowledged for preventing replay attacks. A very low-complexity and space-efficient implementation of such a data structure is a zero-initialized bit vector. Setting bit k in the bit vector signifies that the kth MSG of a certain flow is acknowledged, and, thus, future replays of b_k can be ignored. Specifically, a UE checks whether MSG k of the indicated flow has already been acknowledged (kth bit set), and if yes, discards it. The effectiveness of our replay protection mechanism in shown Section V.

Flow Authentication. The Merkle tree assures that all b_k can be authenticated to a single value, that is, the root H which serves as a flow identifier. Intermediate UEs can validate that b_k belongs to H by traversing the tree from b_k to the root H' using intermediate tree nodes f_k and checking that $H' = H$. The flow authentication procedure has been described in [11] and assures that only MSGs belonging to the flow will be forwarded.

In-Network Flow Authenticator Compression. The size of the flow authenticator f_k has a drastic impact on the protocol overhead. f_k grows linearly with the tree height l. In previous works [11], [13], [19], all sibling nodes in the tree from the leaf to the root (tree nodes $x_1, \ldots, x_l$ in Fig. 3) are included in each message. For large trees, this naïve approach generates significant overhead. For instance, a tree of height $l = 10$ allowing to send $2^{10} = 1024$ MSGs under that flow) requires the header to include 10 hash values for f_k. In absolute terms, these results in $10 \times 32 = 320$ bytes *per MSG* when using a contemporary hash function with a 32-byte output.

SEMUD employs a more efficient method: SEMUD UEs incrementally construct the Merkle tree as they receive new b_k and f_k values (note that intermediate nodes cannot construct the entire tree from n since they do not possess K_{sd}). Starting from the second received MSG, $l' < l$ new tree nodes are required to authenticate the flow. The idea is visualized in Fig. 4. In a stable network, the lower bound average of l' is *constant* with $(2^l - 1)/2^l < 1$ which leads to a 10-fold overhead reduction compared to the example above.

In order to devise a practical distributed algorithm to calculate l', nodes need to keep track of the current Merkle tree state of their neighbors. SEMUD UEs do this by leveraging the ACKs received from their neighbors: when receiving a_k from neighbor h, a UE knows that h has the kth leaf of the Merkle tree, as well as the authenticated path from this leaf to the root. Otherwise, h would have been unable to authenticate and forward the kth MSG in the first place. To determine minimal l', i. e., the shortest possible flow authenticator length for which the next-hop node will still be able to authenticate b_k, we use Algorithm 1. For broadcast MSGs, we set l' to the maximum among all neighbors, i. e., $l' = \max_h l'_h$ with l'_h being the minimal flow authenticator length for neighbor h.

This scheme assures that (i) a UE can always authenticate any MSG received from another correct UE; and (ii) the transmitted flow authenticator does not convey redundant information, i. e., it is exactly as long as it needs to be for minimal MSG overhead. Note that our scheme is agnostic to packet loss and packet reordering.

Automatic Repeat Request. In the rare case that a UE is unable to authenticate the flow because f_k is too short due to lost state, it may "bounce" the MSG back to the sender with the ARQ flag set to request for the complete f_k. The receiving UE then removes the flag and returns the complete f_k of length l to the requester if possible. Otherwise, it drops the MSG.

D. Message Reception

In addition to verification, the destination UE checks the MSG's icv. MSGs with an invalid icv are discarded. For the first MSG of a flow (SYN flag set), the destination locally computes the full Merkle tree as described in Section III-A. For every MSG, the destination selects a_k from the Merkle tree and generates the appropriate ACK (Eq. (2)) which consists of the message digest m and the ACK authenticator a_k. The ACK is then returned to the sender.

$$ACK = \langle m, a_k \rangle \tag{2}$$

E. ACK Handling

ACKs are primarily meant for secure proofs of delivery used to update the neighbor reliability metrics. Upon ACK reception, a UE calculates $b_k = hash\,(a_k)$ and checks whether the ACK belongs to a valid MSG, i. e., whether any MSG with m and b_k has been forwarded before. If not, the ACK

Algorithm 1 Minimal Flow Authenticator Length

```
function MINAUTH(k, l, h)
    l' ← 0
    while l' < l do
        k_left ← (k ⊕ (1 ≪ l)) ∧ (−1 ≪ l)
        k_right ← k_left + (1 ≪ l) − 1
        for k' ∈ [k_left, k_right] do
            if h has acknowledged k' then
                return l'
        l' ← l' + 1
    return l
```

TABLE I: Computation time in μs of several cryptographic algorithms on various platforms for 1024-byte strings averaged over 10 000 runs. Ed25519 is a state-of-the-art elliptic-curve signature scheme and included as a reference.

Class	Algorithm	ALIX	APU	Nexus	Mac
$hash(\cdot)$	SHA-256	184	36	18	6
	Blake2b	167	8	29	3
$prf(\cdot,\cdot)$	XSalsa20/20	97	12	12	5
$tag(\cdot,\cdot)$	SipHash-2-4	66	4	8	1
	HMAC-SHA-512	417	35	92	6
	Ed25519 (verify)	8761	1479	815	168

is dropped. Otherwise, and if the sending UE matches the previous forwarding decision, the reliability metric for the sending UE is increased. The ACK is then forwarded to the neighbors from which the UE received copies of the corresponding MSG. In addition, a valid ACK updates the bit vector used for duplicate detection and neighbor Merkle tree state as described in Section III-C.

IV. Implementation

We choose the Click modular router [20] for SEMUD implementation to allow for a realistic evaluation on both real hardware (Linux, Android) and simulation (ns-3). In this section, we discuss suitable candidate functions for SEMUD's crypto primitives and devise a practical link-local broadcast authentication scheme based on symmetric cryptography.

A. Reference Platforms

We evaluate our SEMUD on heterogeneous platforms with different CPU architectures, processing capabilities, memory configurations (256 MB to 16 GB RAM), and operating systems (Debian Linux, Android 6, macOS 10.11). In particular, these are: *ALIX* [21], *APU* [22], LG *Nexus* 5, and *MacBook* Pro (early 2015).

B. Cryptographic Primitives

The choice of efficient cryptographic primitives is imperative for any practical communication protocol. In SEMUD, cryptographic operations consume the longest processing time during packet forwarding and constitute the major portion of the communication overhead. Our implementation relies on primitives provided by the lightweight, cross-platform libsodium (v1.0.11) [23]. A performance comparison between different candidate algorithms on our reference platforms is shown in Table I. The table also gives an intuition on why public key crypto is unsuitable to be used on a per-MSG basis: the cumulated forwarding delay would be unacceptably large. We select SEMUD's cryptographic primitives as follows:

- $hash(\cdot)$ is implemented as Blake2b with an output size of 16 bytes. Note that the hash function used to construct the Merkle tree does not need to be collision resistant but only preimage and second-preimage resistant.[1] Hence, a 128-bit security margin is sufficiently large.
- $prf(K_{sd}, n)$ is implemented as XSalsa20/20, a stream cipher using 256-bit keys and 192-bit nonces.
- $tag(K_{sd}, \cdot)$ is implemented as SipHash-2-4 [24], which generates small 8-byte authentication tags for short-input (order of kilobytes) messages using a shared secret K_{sd}.

[1] *Collision resistance*: given $hash(\cdot)$, it is hard to find x and x' such that $hash(x) = hash(x')$. *Preimage resistance*: given y, it is hard to find x such that $hash(x) = y$. *Second-preimage resistance*: given x, it is hard to find $x' \neq x$ such that $hash(x) = hash(x')$.

C. Practical One-hop Broadcast Authentication

SEMUD requires neighbor-to-neighbor communication to be authenticated to prevent blackmailing and Sybil attacks. The 3GPP proposed method for secure one-to-one (unicast) communication relies on direct session key exchange between two UEs [8]. Authenticated communication can then be achieved using MACs. For one-to-many or one-to-all (broadcast) communication, the 3GPP proposes an infrastructure-supported scheme which establishes a group key. However, this proposal is unsuitable for public safety-related operations where we assume that the infrastructure might be unavailable. Other cryptographic methods to authenticate broadcast communication are either based on digital signatures or on TESLA [25], which is based on symmetric-key cryptography and delayed key disclosure to achieve asymmetry. We deem both approaches impractical since digital signatures are computationally expensive; and TESLA requires time synchronization between all nodes, and introduces authentication delay which would impede SEMUD's reactiveness to route changes.

The small output size of SipHash enables us to implement a one-hop broadcast authentication scheme based on symmetric-key cryptography without TESLA's deficiencies: a forwarding UE computes authentication tags for each neighbor $h \in \mathbb{F}$ (excluding the sender) and appends all of them to the MSG. A receiving node then tries to authenticate every tag. If any of them succeeds, the MSG is processed, and otherwise discarded. The expected number of SipHash calculations at a receiving node is $|\mathbb{F}|/2$, the communication overhead is $|\mathbb{F}| \times |tag(\cdot,\cdot)|$. We argue that this scheme is practical since (i) the number of neighbors is typically low compared to the total number of nodes in the network (which is what TESLA was designed for), so the communication and computational overhead for transmitting and verifying all tags remains low on average; and (ii) broadcasts are used for route exploration which rarely occurs in established communication flows.

TABLE II: Simulation settings

	Parameter	Value
Network	Number of nodes	100
	Fraction of attackers	0, 10, 20, 30, 40, 50 %
	Dimensions	$3000 \times 3000\,\mathrm{m}^2$
	Random waypoint mobility	$v = 3$ km/h
	Propagation loss model	Free-space path loss
Traffic	Concurrent Flows	10
	Payload size and MSG rate	128 bytes at $2\,\mathrm{s}^{-1}$
Protocol	Merkle tree size 2^l	128 (flow lifetime 64 s)
	Beaconing	rate $4\,\mathrm{s}^{-1}$, timeout 1 s

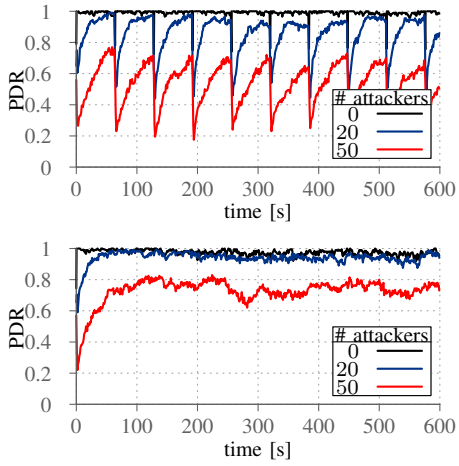

Fig. 5: *Blackhole attack*: reliability of Castor (*top*) and SEMUD (*bottom*).

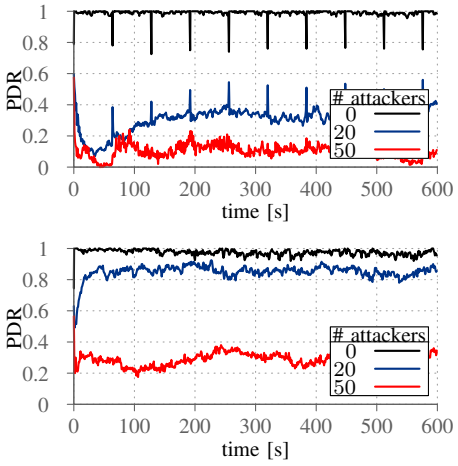

Fig. 6: *Combined replay and blackhole attack*: reliability of Castor (*top*) and SEMUD (*bottom*).

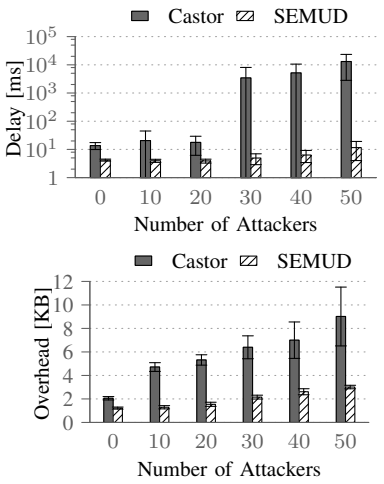

Fig. 7: *Combined replay and blackhole attack*: delay and overhead.

V. Evaluation

In this section, we assess the performance of SEMUD via ns-3 simulations. Moreover, we show the energy and throughput performance on real devices. Due to lack of secure multi-hop D2D schemes, we opted to benchmark SEMUD against Castor [11], the state-of-the-art in MANET security. Castor has been shown to outperform contemporary MANET security schemes such as SEAD [13] and Sprout [26]. We use ns-3 simulations to expose both schemes to two hard-to-detect attacks by powerful adversaries, i.e., blackhole and combined replay and blackhole attack, and analyze the schemes' performance such malicious scenarios. However, the simulations do not provide several important metrics such as energy consumption and realistic throughput measurement due to cryptographic operations on a real device. To this aim, we evaluate SEMUD against Castor using real devices.

We are primarily interested in four metrics: (i) packet delivery ratio (PDR), (ii) delay, (iii) protocol overhead, and (iv) energy consumption. The duration of each simulation/measurement is 10 minutes. The bar plot figures report both the average values and 95% confidence intervals of 10 differently-seeded runs. Our energy measurements assume out-band D2D which is expected to be the first D2D deployment due to less regulatory complications [27], [28]. Nonetheless, SEMUD's design relies on the ProSe architecture and is independent of the D2D mode. In our evaluation, we assume that the supporting infrastructure is not available. Note that 3GPP still mandates the UEs who wish to use D2D services to have subscribed to ProSe at some time in the past. The security-related evaluation is performed in the ns-3 network simulator with Click implementations of SEMUD and Castor. Detailed simulation settings for ns-3 are provided in Table II.

A. Blackhole Attack

The *blackhole* attacker drops messages in order to cause maximum harm to network operation. First, it plays along during route exploration by forwarding broadcast messages to place itself on an active path. Then, it drops all unicast messages during route exploitation to disrupt ongoing communication. In Fig. 5, we observe that SEMUD exhibits a more stable behavior than Castor. The PDRs of both schemes have an almost identical increasing trend during the first 64 seconds of the simulation. In this time frame, both schemes are gradually identifying and circumventing the blackhole attackers according to the procedure defined in Section III. While SEMUD approaches a rather stable PDR, Castor's PDR drops periodically due to *flow restarts*. Flow restarts occur if the source ran out of MSG identifiers b_k, i. e., leaves of the Merkle tree, for the current flow. In this case, the source UE needs to construct a new Merkle tree to continue the communication. Since the old and the new flow identifiers are cryptographically unrelated, Castor re-explores the path towards the destination. Thereby, Castor UEs also "forget" the compromised paths forcing them to re-identify the attackers. The periodicity (64 seconds) of the drops are equal to the flow lifetime (2^l/ MSG rate). Note that the larger is the Merkle tree, the higher is the protocol overhead in terms of bandwidth and processing time. SEMUD circumvents the issue of flow restarts by *opportunistic flow initialization*: instead of re-exploring the path, SEMUD UEs try to leverage their prior network knowledge (i. e., reliability metrics).

B. Combined Replay and Blackhole Attack

In this section, we expose UEs to even stronger attackers, which, in addition to selectively dropping messages, inject expired MSGs and ACKs to reinforce their appearance as reliable forwarders. We first sketch the combined replay and blackhole attacker $\mathcal{M}$ attemping to jeopardize the communication between two UEs $\mathcal{A}$ and $\mathcal{B}$: First, $\mathcal{M}$ overhears and records any valid MSG–ACK pair k of some flow H between $\mathcal{A}$ and $\mathcal{B}$. Then, $\mathcal{M}$ replays (i. e., broadcasts) MSG k and ACK k shortly after one another at an interval of T_{rep} (after an initial delay of T_{rep}). The attacker chooses T_{rep} such that it is larger than the ACK timeout, i. e., $T_{\text{rep}} > T_{\text{ACK}}$. To

ensure this, the attacker conservatively sets T_{rep} uniformly from 1000 ± 300 ms for each pair. The jitter (± 300 ms) is introduced to avoid synchronization between attacking nodes. Each attacker replays at most 5 pairs per second.

Fig. 6 demonstrates that SEMUD significantly outperforms Castor in this scenario: with 20 % attackers, SEMUD's PDR remains above 80 %, while Castor's is below 40 %. This is due to the fact that SEMUD employs a more effective replay protection mechanism than Castor. Since Castor UEs do not keep the state for MSG k after the ACK timeout, they can be tricked into accepting the same MSG k for flow H again. Since the kth ACK has potentially been disclosed by the destination, the attacker can directly acknowledge its own replayed MSG. As a result, intermediate UEs increase the reliability metric for the attacker. Thus, Castor UEs are likely to forward new MSGs for flow H to the attacker. In Section IV-C, the PDR peaks at 20 attackers occur at flow restarts since Castor UEs are paralyzed by the attackers and flooding becomes the most reliable option. Note that the achieved PDR during flow restarts in this scenario never exceeds those observed in Section IV-C. Finally, when the fraction of attackers reaches 50 %, neither scheme delivers acceptable results. With such a high density of attackers, the medium is simply jammed by replayed MSG and ACKs.

Fig. 7 shows that Castor collapses once the fraction of attackers reaches 30 %: the delay skyrockets above 3 seconds. This occurs as Castor UEs are themselves forwarding replayed MSGs, thus, amplifying the effective jamming impact of the attackers. The result is DoS caused by exceeding the network capacity. However, SEMUD maintains a delay below 11 ms due to ignoring the attackers and, thus, containing their impact radius to a single hop. We also observe that Castor's protocol overhead increases tremendously since Castor UEs unwantedly amplify the attack. In contrast, the attack has an insignificant impact on SEMUD's protocol overhead.

C. Achievable Throughput

Since SEMUD applies cryptographic operations per MSG, we are concerned about data throughput. Table III shows that the choice of lightweight symmetric-key cryptography and efficient hash algorithms allow competitive data rates even on dated platforms. For example, the Nexus 5 can handle almost 10 000 MSGs per second. In the next section, we also shed light on the energy efficiency of SEMUD on the same platform.

D. Smartphone Energy Consumption

Since the SEMUDs ProSe Application performs cryptographic operations on battery-powered devices, it is critical to ensure that such operations do not incur significant energy consumption. In this experiment, we assess SEMUD's impact on battery lifetime of an LG Nexus 5. The test setup consists of two smartphones, one of which acts as a traffic generator, the other as a receiver. To measure the energy consumption, we use the Monsoon power monitor which connects to the battery contacts of the smartphone [29]. The traffic generation (100 messages per second) runs for 5 minutes and starts at the 3-minute mark. We measure the average energy consumption over 10 minutes. We show the results in Fig. 8. The idle power consumption comprises Android background tasks and the active wireless ad hoc connection. During transmission, we observe an increase of 100 mW which consists of (i) wireless transmissions and (ii) SEMUD message processing. Note that the power consumed by SEMUD during 5 minutes of active transmission amounts to only 30 seconds of screen power consumption (1000 mW). The results should be similar for other phone models [30]. During our measurements, the screen is turned off. However, the screen turns on at the beginning of the measurements due to the power analyzer disconnecting the USB port. This causes the power consumption peak during the first seconds.

TABLE III: Achievable source throughput of our Click implementation on various platforms with a payload size of 1024 bytes per MSG. Goodput considers only user payload.

METRIC	ALIX	APU	NEXUS	MACBOOK
Throughput (*MSG/s*)	3641	19840	9524	149525
Goodput (*Mbit/s*)	29.01	158.03	75.84	1190.98

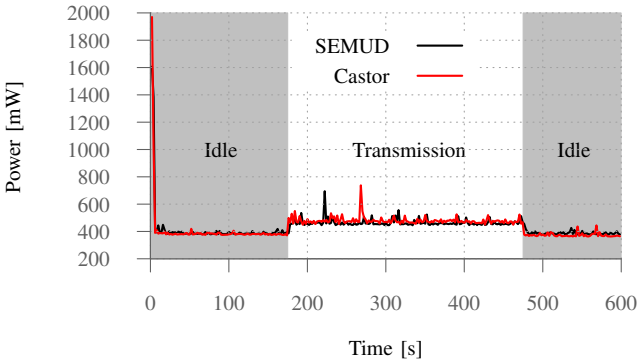

Fig. 8: SEMUD power consumption on a Nexus 5.

VI. RELATED WORK

While the literature on D2D communications is abundant [31], the majority of articles focuses on issues such as scheduling, interference management, power allocation, and architectural enhancement. D2D-specific security aspects have received little attention so far and mainly focused on authentication and key management [6], [32], [33]. However, there is no work on secure multi-hop D2D communication. As a result, we focus on related schemes in MANETs, which shares some characteristics of D2D-enabled networks. Early work in this field are less comprehensive with respect to security attack resilience or practicality consideration. SAODV [34] and SRP [35] are both vulnerable to the tunneling attack (a weaker form of the wormhole attack). ARAN [36] and SEAD [13] rely on expensive public key cryptography. Ariadne [37] successfully routes around multiple colluding malicious nodes, but requires security associations between all nodes on the path which causes scalability issues for key distribution. All above schemes only secure control messages and disregard

the security of user data which make them vulnerable to packet dropping. Sprout [26] introduces probabilistic path probing to also secure data transmission using signed end-to-end acknowledgments. Castor [11] also uses end-to-end acknowledgments, but achieves higher resilience against sophisticated attacks such as blackholes and wormholes by incorporating an implicit and independent route discovery. However, compared to SEMUD communication, Castor is susceptible to replay attacks, suffers from flow restarts, and exhibits a much larger per-message overhead. All of the above proposals lack considerations for practical deployment, i. e., problems such as trust establishment or key distribution are left out and taken for granted. Our use of the ProSe architecture provides a working solution to this problem.

VII. Conclusion

The provisioning of robust and secure multi-hop D2D communication is crucial for safety-critical applications in 5G networks. In this article, we describe the design of SEMUD, which offers the first standard-compliant solution for this purpose. SEMUD leverages ProSe infrastructure components for bootstrapping; and as a trust anchor during infrastructure-based operation. Its designation, however, is to operate in the absence of infrastructure in multi-hop D2D settings. For these settings, SEMUD offers strong security and robustness guarantees, even under adversarial conditions. We have shown the feasibility and practicability of our approach by means of a prototypical implementation for the ns-3 simulator as well as mobile end systems. In the absence of competing multi-hop D2D security solutions, we benchmark SEMUD against Castor, a state-of-the-art secure MANET scheme. Our simulation results reveal that SEMUD outperforms Castor under several powerful and hard-to-detect attacks. In particular, we show that SEMUD increases the PDR by up to a factor of 3 and reduces delay at least by a factor of 2.6. The measurement results on Nexus 5 smartphones indicate that SEMUD incurs low overhead allowing high throughput and low energy consumption. Finally, we make our source code publicly available [38].

Acknowledgment

This work has been co-funded by the LOEWE initiative (Hessen, Germany) within the NICER project, and by the German BMBF within CRISP.

References

[1] J. M. B. da Silva, G. Fodor, and T. F. Maciel, "Performance analysis of network-assisted two-hop D2D communications," in *IEEE Globecom Workshops*, 2014.

[2] H. Nishiyama, M. Ito, and N. Kato, "Relay-by-Smartphone: Realizing Multihop Device-to-Device Communications," *IEEE Commun. Mag.*, vol. 52, no. 4, 2014.

[3] Y. Wu *et al.*, "Device-to-device (D2D) meets LTE-unlicensed," *IEEE Commun. Mag.*, 2016.

[4] B. Badic *et al.*, *Rolling Out 5G: Use Cases, Applications, and Technology Solutions*. Springer US, 2016.

[5] A. Shaik *et al.*, "Practical Attacks Against Privacy and Availability in 4G/LTE Mobile Communication Systems," in *NDSS*, 2015.

[6] M. Wang and Z. Yan, "A survey on security in D2D communications," *Mobile Networks and Applications*, 2016.

[7] 3GPP, "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Proximity-based services (ProSe); Stage 2 (Release 13)," *TR 23.303 V13.0.0*, 2015.

[8] ——, "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Proximity-based Services (ProSe); Security aspects (Release 13)," *TR 33.303 V13.3.0*, 2016.

[9] T. Shu and M. Krunz, "Privacy-preserving and truthful detection of packet dropping attacks in wireless ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 14, no. 4, 2015.

[10] S. Paris *et al.*, "Cross-layer metrics for reliable routing in wireless mesh networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 3, 2013.

[11] W. Galuba *et al.*, "Castor: Scalable secure routing for ad hoc networks," in *IEEE INFOCOM*, 2010.

[12] A. Asadi, P. Jacko, and V. Mancuso, "Modeling multi-mode D2D communications in LTE," *arXiv preprint arXiv:1405.6689*, 2014.

[13] Y.-C. Hu, D. B. Johnson, and A. Perrig, "SEAD: secure efficient distance vector routing for mobile wireless ad hoc networks," *Ad Hoc Networks*, 2003.

[14] B. Hu and H. Gharavi, "Smart grid mesh network security using dynamic key distribution with merkle tree 4-way handshaking," *IEEE Transactions on Smart Grid*, vol. 5, no. 2, 2014.

[15] A. Malhotra *et al.*, "Attacking the Network Time Protocol," in *USENIX NDSS*, 2016.

[16] P. Papadimitratos and A. Jovanovic, "GNSS-based positioning: Attacks and countermeasures," in *IEEE MILCOM*, 2008.

[17] P. Papadimitratos and Z. J. Haas, "Secure Data Communication in Mobile Ad Hoc Networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, 2006.

[18] M. Schmittner and M. Hollick, "Xcastor: Secure and Scalable Group Communication in Ad Hoc Networks," in *IEEE WoWMoM*, 2016.

[19] M. Islam and M. Hamid, "SHWMP: A secure hybrid wireless mesh protocol for ieee 802.11s wireless mesh networks," *Transactions on Computational Science VI*, 2009.

[20] E. Kohler *et al.*, "The Click Modular Router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, 2000.

[21] ALIX platform. Online: http://www.pcengines.ch/alix.htm

[22] APU platform. Online: http://www.pcengines.ch/apu.htm

[23] "The Sodium crypto library (libsodium)." Online: https://libsodium.org

[24] J. P. Aumasson and D. J. Bernstein, "SipHash: A fast short-input PRF," in *LNCS*, vol. 7668, 2012.

[25] A. Perrig *et al.*, "Efficient Authentication and Signing of Multicast Streams over Lossy Channels," in *IEEE S&P*, 2000.

[26] J. Eriksson, M. Faloutsos, and S. V. Krishnamurthy, "Routing amid colluding attackers," in *IEEE International Conference on Network Protocols*, 2007.

[27] A. Asadi, V. Mancuso, and G. Rohit, "An SDR-based Experimental Study of Outband D2D Communications," in *IEEE INFOCOM*, 2016.

[28] S. Andreev *et al.*, "Cellular Traffic Offloading onto Network-Assisted Device-to-Device Connections," *IEEE Commun. Mag.*, 2014.

[29] M. Schulz, "Preparation of a Nexus 5 Android Smartphone for Power Analysis," Tech. Rep., 2016.

[30] A. Rice and S. Hay, "Measuring mobile phone energy consumption for 802.11 wireless networking," *Pervasive Mob. Comput.*, vol. 6, no. 6, 2010.

[31] A. Asadi, Q. Wang, and V. Mancuso, "A Survey on Device-to-Device Communication in Cellular Networks," *IEEE Communications Surveys & Tutorials*, 2014.

[32] W. Shen *et al.*, "Secure key establishment for device-to-device communications," in *IEEE GLOBECOM*, 2014.

[33] A. Zhang *et al.*, "SeDS: Secure Data Sharing Strategy for D2D Communication in LTE-Advanced Networks," *IEEE Trans. Veh. Technol.*, 2016.

[34] M. G. Zapata and N. Asokan, "Securing ad hoc routing protocols," in *ACM Workshop on Wireless Security*, 2002.

[35] P. Papadimitratos and Z. J. Haas, "Secure routing for mobile ad hoc networks," in *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference*, 2002.

[36] K. Sanzgiri *et al.*, "A Secure Routing Protocol for Ad Hoc Networks," in *IEEE ICNP*, 2002.

[37] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: A Secure On-demand Routing Protocol for Ad Hoc Networks," *Wireless Networks*, vol. 11, no. 1-2, 2005.

[38] "SEMUD source code." Online: https://seemoo.de/semud