Plug-and-Play IP Security:
Anonymity Infrastructure Instead of PKI

Yossi Gilad and Amir Herzberg

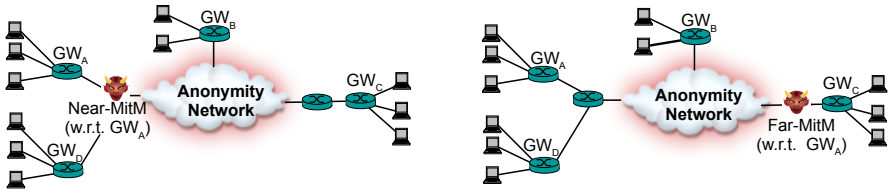
Department of Computer Science, Bar Ilan University
mail@yossigilad.com amir.herzberg@gmail.com

Abstract. We present the *Plug-and-Play IP Security (PnP-IPsec)* protocol. PnP-IPsec automatically establishes IPsec security associations between gateways, avoiding the need for manual administration and coordination between gateways, and the dependency on IPsec public key certificates - the two problems which are widely believed to have limited the use of IPsec mostly to intra-organization communication.
PnP-IPsec builds on *Self-validated Public Data Distribution (SvPDD)*, a protocol that we present to establish secure connections between remote peers/networks, without depending on pre-distributed keys or certification infrastructure. Instead, SvPDD uses available anonymous communication infrastructures such as Tor, which we show to allow detection of MitM attacker interfering with communication. SvPDD may also be used in other scenarios lacking secure public key distribution, such as the initial connection to an SSH server.
We provide an open-source implementation of PnP-IPsec and SvPDD, and show that the resulting system is practical and secure.

1 Introduction

Consider two Internet users, Alice and Bob. Alice wants to communicate securely, and possibly anonymously, with Bob. For anonymity, Alice may use an anonymity service, such as the *Tor* network of relays [5]. However, Alice also wants to encrypt her messages to Bob; how can she obtain securely Bob's public key?

The standard answer is that Alice will send a request to Bob and receive back his public key, certified by a trusted *Certificate Authority (CA)* [12], like in normal use of SSL/TLS, e.g., by browsers; if anonymity is desired, all communication would be via the anonymity service, e.g., Tor. However, this does not apply to the IP-security protocol (IPsec) [15], where traditional certificates are less appropriate, and which requires configuration (of security policies, network blocks, etc.). Furthermore, users may prefer complementary or alternative mechanisms to trusting a CA, e.g., due to several incidents where CAs authentication mechanisms were broken and false certificates were issued: CAs have been compromised, e.g., [4], and used insecure cryptographic primitives [23]. Can Alice securely receive Bob's public key, without depending on a trusted CA for authentication? Can she take advantage of IPsec, if supported by Bob?



(a) Near-MitM Attacker. Communication between $\mathsf{GW_A}$ and *all* other peers routes via the attacker.

(b) Far-MitM Attacker. Only communication between $\mathsf{GW_A}$ and $\mathsf{GW_C}$ routes via the attacker.

Fig. 1. Types of MitM attackers with respect to $\mathsf{GW_A}$.

In this work we show that this is possible. We first present Self-validated Public Data Distribution (SvPDD), which provides public key distribution using an anonymity service, instead of relying on authentication and certification of Bob by a trusted CA. SvPDD can use an existing anonymity service such as Tor, the largest public anonymity network, as we do in our prototype implementation.

The basic idea of SvPDD is simple: Bob will periodically *self-validate* that communication to and from himself is not tampered with, by sending *to himself* anonymized requests for his public key, and validating that his responses arrive correctly (with correct public key) and in timely fashion. Any tampering by a MitM attacker with the response (public key) would be detected by Bob. Similarly, Alice will use the anonymity network to send self-addressed 'requests'; a MitM trying to block Alice's communication will not be able to distinguish between this 'self-test' communication and 'real' communication between Alice and Bob, and hence Alice will detect any tampering.

SvPDD detects when a MitM attacker disrupts communication as well as points-out the attacker's *location*. We classify MitM attackers with respect to a particular party $\mathcal{P}$ to either of two types, illustrated in Figure 1:

Near-MitM who can manipulate communication between $\mathcal{P}$ and a significant portion the network. This attacker will usually be en-route between $\mathcal{P}$ to the anonymity network.

Far-MitM who can manipulate communication between $\mathcal{P}$ and few remote peers. This attacker will usually be 'near' with respect to those peers.

This property is significant; a system administrator cannot do much about a 'far' MitM attacker disrupting communication with some peer and may ignore such warning, but an alert about a 'near' MitM attacker is likely to result in immediate corrective actions (such as changing ISP or scanning for malware).

SvPDD seems especially beneficial to facilitate adoption and deployment of *IPsec*, the standard protocol for cryptographically-protecting IP traffic. IPsec is a mature, well-validated protocol providing strong security guarantees. In particular, IPsec provides defenses against Denial of Service (DoS) attacks, while the main alternatives, SSL and TLS, run over TCP, and hence are vulnerable to TCP's DoS attacks such as SYN flooding [6] and Ack-Storm [1] (although note

that IPsec should also be implemented correctly to avoid DoS vulnerabilities, see [11]). IPsec is implemented in most operating systems and in many devices (it is even part of IPv6 specification). However, actual use of IPsec is very limited; the main reason seems to be the difficulty in establishing IPsec connections, which normally require manual establishment of keys. SvPDD provides an alternative, allowing secure and completely-automated establishment of IPsec keys between peers, without requiring (rarely-available) IP-address based certificates.

There is another challenge to the deployment of IPsec: the need to coordinate its use among peers. Even if all IPsec peers had appropriate public-key certificates from a trusted CA, in order for IPsec to be deployed between two peers, each peer must be aware of the deployment at the other end, by an appropriate security-policy rule setup by the administrator. Coordination is even more challenging to support IPsec's *tunnel mode*, where an IPsec gateway machine protects an entire network; here, the security-policy rule must specify the network block(s) connected via the given peer (network gateway).

To completely address the IPsec deployment challenges, we present the *Plug-and-Play IP Security (PnP-IPsec)* protocol, built on top of SvPDD. PnP-IPsec automatically establishes IPsec security associations between networks (and/or hosts); see the layering of PnP-IPsec and SvPDD in Figure 2. PnP-IPsec adds two functions to SvPDD: (1) automated detection of remote peers, including handling of scenarios where there are multiple PnP-IPsec gateways en route to the destination; and (2) validation of the address block protected by the remote gateway. In order to establish a secure IPsec connection between two networks, all that is required is for each of the networks to independently run PnP-IPsec; all the rest is done automatically by PnP-IPsec.

1.1 Our Contribution

We present two protocols, SvPDD and PnP-IPsec. SvPDD uses an available anonymization service such as Tor to establish secure public keys between peers, without requiring off-path key distribution or certification authorities. This provides alternative means for validation of public keys, for protocols and systems where appropriate public-key certificates are unavailable.

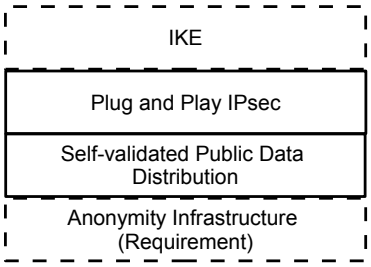


Fig. 2. The layering of our protocols (boxed with solid lines) and related protocols (boxed with dashed lines).

PnP-IPsec, built on top of SvPDD, allows automated IPsec tunnel establishment, without requiring coordinated administration or key distribution infrastructure. PnP-IPsec automatically detects the existence of a remote PnP-IPsec gateway, obtains its public key and network block, and validates that the remote gateway indeed controls that network block.

SvPDD and PnP-IPsec provide the following defense against MitM attackers:

- A far MitM attacker w.r.t. both peers cannot interfere with the protocol.
- A near-MitM with respect to one of the peers may interfere with the protocol, but in this case it will be detected by the administrator of that peer.

We provide an open-source implementation for our protocols (see Section 7) and hope that this will increase deployment of IPsec.

SvPDD is not limited to IPsec, and our implementation may be integrated into other protocols, such as SSH, in order to cope with a MitM attacker during the initial setup (when the user learns the server's unauthenticated public key), or even TLS/SSL, to provide additional means to obtain and validate the public key (protect users against CA authentication failures).

Lastly, this paper has the conceptual contribution of showing how anonymity infrastructure can be used in lieu of PKI, to establish security between arbitrary peers, without common administration, pre-shared keys or CAs.

1.2 Related Work

Ishai et al. [13] presented a theoretical study of how two parties may use a shared anonymous broadcast medium, to establish a shared secret key between them; the two parties work in coordinated manner, which in practice implies, they could have also established keys while coordinating, hence their work is not of much practical impact. However, their work does provide some of the concepts used and extended in our work, where we establish keys between arbitrary parties, without assuming any coordination between them in advance. Hence, our work extends their conceptual contribution, and shows that the basic idea of using anonymity to establish security can also have practical implications.

There have been multiple efforts to simplify deployment of cryptographic protocols by automating their setup, without certification authorities or coordinated management; we discuss these efforts below.

Several protocols, such as SSH [27] and BTNS [24, 26], are based on the *Leap of Faith (LoF)* approach (also called 'Trust On First Use'). In LoF, public keys are exchanged without any validation during the first connection, and later used (assuming the initially-exchanged public keys were correct); SSH applications also display the public key to the users, allowing users to use off-path validation of the public key (but few do). LoF protocols assume a handicap of the MitM attacker, i.e., that he does not impersonate during the initial handshake; in contrast to these works, SvPDD and PnP-IPsec do not assume this limitation.

A notable effort for mitigating the need for coordinate deployment of IPsec is by the FreeS/WAN project [9], who attempted ([22]) to create an *opportunistic* version of IKE, as documented in [19, 20]. The specification requires the network administrator to place a reverse DNS record mapping to the network's gateway and public key. The initiator retrieves the DNS record and uses the fetched configuration (gateway address and public key) to start the IKE negotiation. However, using [20] requires configuration of the reverse DNS tree, which is complex, and furthermore allows only one level of gateways - typically, by an ISP

or a large organization; it does not allow multiple gateways, or protection of small networks and individual hosts (who do not control the reverse-DNS records).

Perspectives [25] and Convergence [17] are proposals for web-server public key validation mechanisms, to replace or complement the existing certificates (issued by CAs trusted by the browsers). Both rely on the use of a set of trusted 'notary' servers, which collect (and potentially cache) the public keys for the users. The idea is that a MitM near the client is not en route between most of the notaries and the server, allowing the client to learn the keys from the notaries (according to their majority). SvPDD performs a similar function to these proposals, with two advantages: (1) SvPDD does not require establishment and maintenance of a new infrastructure of notaries, and instead leverages an existing, general-purpose, anonymity infrastructure (Tor), which has many users and handles high traffic rates, compared to which the traffic generated by our protocols is negligible (see [18]); and (2) SvPDD provides better security to the users by not requiring them to trust new entities for authentication, and only to trust the anonymity network to anonymize their requests.

Double-Check [2] shows how one can validate self-signed certificates by accessing the server from various locations, suggesting Tor as an available proxy infrastructure. Double-Check helps against a MitM attacker that controls some of the routes to the server, but fails if attacker controls all (or most) of the routes *from the client* or *to the server*. In contrast, SvPDD utilizes anonymity, and suggests the concept of self-validation. SvPDD provides the same benefits as Double-Check, and in addition, using self-validation, SvPDD detects and provides a clear indication when an attacker controls all (or most) of the routes near the client or near the server.

PnP-IPsec shares some aspects with a previous work of ours, LOT [7], an opportunistic tunneling protocol for establishing credentials between two arbitrary networks in order to detect and block spoofed packets. However, there are substantial differences. First, LOT was designed to secure against off-path (non-eavesdropping) rather than MitM attackers. Second, LOT creates hop-by-hop tunnels, decapsulating and re-encapsulating information at every node on the path; this property is avoided in PnP-IPsec, which establishes gateway-to-gateway IPsec tunnels.

This is an extended version of our conference publication [8].

2 SvPDD: Model and Security Requirements

SvPDD runs on two peers, a querier and a responder, without coordinated management or common public key infrastructure. The basic goal is that the querier will learn the responder's response for his query; however, clearly if there is a MitM connecting one of the peers to the network, then the MitM can prevent satisfying this goal simply by blocking all communication between the peers. This section describes the model and security requirements of the SvPDD protocol.

Anonymity Infrastructure. We assume the availability of an anonymity network. Peers can send messages via the anonymity network, hiding the intended recipient; and receive messages from the network, while the sender remains hidden.

Furthermore, we assume that the querier has the public key of the anonymity network, i.e., can send authenticated and encrypted messages to it; this property holds for many anonymity networks, such as Tor [5] and Mix-Nets [21], where the client has a hard-coded copy of the network's public key.

Notice that while the querier sends and receives authenticated content from the anonymity network, he *does not* trust the network to authenticate other peers (in contrast to CAs in the public key infrastructure).

Attacker Model. We consider two types of MitM attackers, defined according to the near-MitM threshold, denoted by δ: a *near-MitM* attacker with respect to a peer $\mathcal{P}$ obtains a message that $\mathcal{P}$ sends or receives from the anonymity network with probability greater than δ; otherwise, the attacker is considered a *far-MitM* with respect to $\mathcal{P}$. If $\mathcal{A}$ *obtains* a message, then he can block it or modify its content (MitM capabilities). We assume that the attacker is either near the querier or responder (but not both), or far with respect to both peers.

Based on the analysis that we present in Section 4, we require that $0 < \delta \leq \frac{1}{8}$; the exact value of δ is a local configuration provided by the system administrator, who essentially sets the threshold for a MitM-alert: the lower that δ is, the more attackers will be classified as 'near-MitM' (in our implementation the default configuration is $\delta = \frac{1}{10}$).

Communication Model. When a peer $\mathcal{P}$ *sends* a message to the anonymity network: if the MitM attacker $\mathcal{A}$ is near $\mathcal{P}$ then he obtains the message, as well as the identity of the sender; otherwise, $\mathcal{A}$ obtains the message and sender's identity with probability δ.

Similarly, when a peer $\mathcal{P}$ *receives* a message from the anonymity network: if $\mathcal{A}$ is near $\mathcal{P}$, then he obtains the message as well as the identity of the recipient; otherwise, $\mathcal{A}$ obtains the message and identity of the recipient with probability δ.

Notice that our communication model is the 'worst-case' scenario, where a near-MitM obtains a message from or to $\mathcal{P}$ with probability 1 (i.e., obtains all such messages), and a far-MitM obtains such a message with probability δ.

Security Requirements. A public data distribution protocol with security parameter n is secure if the following properties hold, except with negligible probability in n:

No False Alert: if $\mathcal{A}$ is far with respect to $\mathcal{P}$, then $\mathcal{P}$ does not alert for MitM.
Authenticity: if neither peer alerts for MitM, then the querier learns the correct response for his query, exactly as sent by the responder.

From these properties follows the *availability* property: if $\mathcal{A}$ is far with respect to both peers, then the querier learns the correct response for his query.

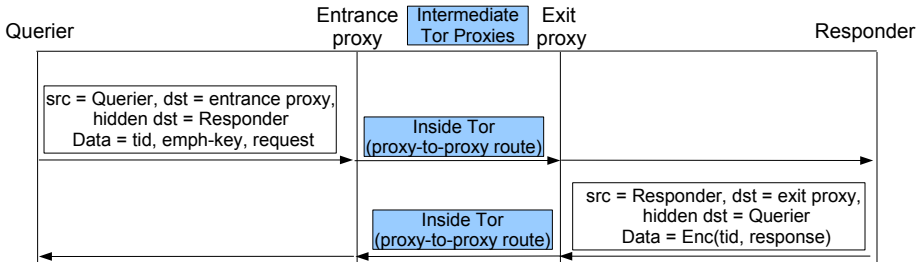


Fig. 3. A Query-Response Transaction over the Tor Anonymity Network.

3 SvPDD: Protocol

In this section we present Self-validated Public Data Distribution (SvPDD), a protocol that allows a querier to retrieve and validate content from a responder and satisfies the security requirements in Section 2.

3.1 The Query-Response Transaction

In an SvPDD *transaction* the querier sends a query for which the responder sends a response, both messages are transmitted via the anonymity network; see illustration in Figure 3. Each transaction belongs to one of two classes:

Peer-to-Peer (p2p). The querier attempts to learn the responder's response.
Self. A 'dummy' transaction, the peer is both the querier and the responder.

Each transaction has a random identifier, denoted by *tid*, which is chosen by the querier and attached to the transaction messages. We refer to a message that belongs to a p2p/self-transaction as a p2p/self-message (respectively).

A peer *can validate* that self-transaction messages were not modified or blocked by a MitM since the peer is both the sender and recipient of messages: he knows 'what he sends' and compares it with 'what he receives'. In order to keep track of self transactions, each peer keeps a global *self-table* that maps the identifiers of self-transactions to their corresponding messages as sent and received (to allow validation), as well as each message's transmission time.

Message Indistinguishability. An important property of SvPDD messages is that two messages of the same type (query or response), but of different classes (p2p and self), are *indistinguishable*. Namely, a MitM attacker who observes the message (that routes via the anonymity network) usually cannot learn the identities of both the sender and recipient, and detect whether they are different (a p2p-message) or the same (a self-message).

The following describes the content of query and response messages:

Query Message. The querier initiates the transaction by sending a query message to the responder. The message specifies a random ephemeral public key that the querier generates[1] and the request from the responder (see Figure 3).

In our model (described in Section 2), the querier has the public key of the anonymity network, and therefore queries are authenticated and encrypted until they leave the anonymity network (to reach their destination). However, a MitM near the responder can observe the clear-text query; in order to satisfy the desired message indistinguishability property, we require that the query is either constant (e.g., all SSH clients specify the same query, for the server's public key) or chosen according to a *fixed* distribution that is independent of the querier's identity or message history.

Response Message. When a responder receives a query message, he replies with a response message. The response specifies the requested data encrypted with the querier's ephemeral key (see Figure 3). Note that we rely on the indistinguishability property of the (probabilistic) encryption scheme [10], hence, a MitM attacker who observes the response cannot learn according to its content whether it is a response for a self or p2p query (unless the MitM modifies the encryption key in the query, risking that the query was 'self').

Transaction Completion. A transaction is *complete* if one of the following conditions is true: either (1) a response was received in context of this transaction, or (2) the query is stale (decided according to its transmission time). In the latter case, we say that the transaction is expired.

3.2 The Query-Response Session

In order to retrieve data from the responder, the querier starts an SvPDD-*session* which is composed of n p2p-transactions (where n is a security parameter). In each transaction in the session, the querier sends the same request (but with a different transaction identifier and ephemeral public key) to the responder. The querier saves a per-session *p2p-table* which maps the transaction identifier (*tid*) to the corresponding query and response (if received).

The querier and responder perform self-transactions in the background, in parallel to ongoing query-response sessions (see details in Section 3.3).

Message Validation. When a peer receives a message, it first checks whether its *tid* field indicates a self-transaction; if yes, then the message is assigned the class 'self' and otherwise the class 'p2p'. The validation process is different for each message class.

If the received message (query or response) is a self-message, then the peer validates that the message was not modified while it was in-transit. If the self-message was modified, then its transaction is marked as 'failed'.

[1] The ElGamal encryption scheme, for example, allows to efficiently generate private and public key-pairs.

In contrast to self-messages, the recipient peer cannot validate the content of p2p-messages. The recipient only validates, in case of a p2p-response, that it belongs to an uncompleted transaction in some session (otherwise the response is discarded).

MitM Detection. Each self-transaction is associated with a result that is either success or failure. The result of a self-transaction is failure if: (1) it expired (see 'transaction completion' in Section 3.1); or (2) the transaction was marked as 'failed' during the message validation process (above). When a self-transaction completes, its result is enqueued in a cyclic, n entry long, history queue (where n is the number of transactions in each session).

If there are at least $3\delta n$ 'failure' results in the party's history queue, where $0 < \delta \leq \frac{1}{8}$ is the near-MitM threshold (see Section 2), then SvPDD *alerts* the local administrator of a near-MitM.

Session Completion. An SvPDD-session completes when all its transactions have completed. The session is then associated with a success or failure result, depending on the responses that were received for the queries in its context: If more than $\frac{1}{2}n$ (i.e., a majority) transactions of that session received an identical response, then the session result is success and that response is returned. Otherwise, the session's result is failure and no response is returned.

Notice that the threshold for a near-MitM alert ($3\delta n$) is lower than that of completing a session in success ($\frac{1}{2}n > 3\delta n$, since $\delta \leq \frac{1}{8}$). In the following section we present a security analysis and show that this property ensures the desired security requirements, defined in Section 2.

3.3 Protocol Execution

In order to retrieve authenticated data from the responder, the querier starts an SvPDD-session. Additionally, SvPDD runs in the background, on both the querier and responder, and initiates self-transactions. SvPDD monitors the results of the n recent self-transactions, and alerts for a MitM in case that $3\delta n$ of them are assigned the 'failure' result.

Self-Transactions Instantiation. SvPDD approximates the rate of p2p-messages and sends self-messages at roughly the same rate. The reasoning is that if the peers send only few self-messages, then a MitM can change arbitrary messages, which are likely to be p2p; in contrast, if the peers send many self-messages, then SvPDD's overhead grows large.

A peer $\mathcal{P}$ instantiates approximately one self-transaction for every p2p-transaction. This is achieved by measuring $r(t)$, the number of new p2p-transactions that $\mathcal{P}$ participates-in during time period t (each period has the same length). During period $t+1$, $\mathcal{P}$ instantiates $r(t)+c$ new self-transactions; where $c \geq 1$ is a constant value, such that even if the rate of new p2p-messages increases during period $t+1$, it is still likely to be less than the number of new self-transactions.

3.4 Instantiation over Tor

One of the advantages of SvPDD is that suitable anonymity infrastructures are already available. In particular, it is possible to instantiate SvPDD over Tor [5], the largest publicly available and well-studied anonymity network. Using Tor, queries and responses route via a *Tor circuit*, which is a chain of proxies (chosen by the querier), see Figure 3. Each transaction is relayed over a different random Tor circuit, such that transactions of the same session cannot be associated together by a MitM observer.

The querier (running the Tor-client software) has the public keys of the Tor proxies, which are used to authenticate and encrypt query messages until they leave the network to reach the responder. This satisfies our assumption on the anonymity network from Section 2.

In Appendix A we describe the Tor network and SvPDD instantiation over it in greater detail.

4 SvPDD: Analysis

In this section we show that SvPDD satisfies the security requirements presented in Section 2.

No False Alert Requirement. A far MitM with respect to a peer $\mathcal{P}$ obtains a message (sent to or from $\mathcal{P}$) with probability δ. Since in every transaction there are two messages (request and response), the probability that the far MitM attacker obtains at least one message of a transaction is no more than 2δ (in this case the attacker can modify or block the message, i.e., corrupt the transaction).

Let the random variable η denote the number of self-transactions, out of the recent n self-transactions, where the far MitM obtains at least one message. The expected value of η is $E[\eta] \leq 2\delta n$. However, the attacker must modify or block at least $3\delta n$ messages of the n recent self-transactions in order to cause a false alert for near-MitM (see SvPDD definition in Section 3.2).

Hoeffding's inequality allows to bound the probability that $\eta \geq 3\delta n$; i.e., that η deviates from its expected value by at least δn, see Equation 1:

$$\Pr[\eta \geq 3\delta n] \leq e^{-2(\delta n)^2} \tag{1}$$

This bound shows that the probability that the far-MitM attacker succeeds in causing a false alert is a negligible function in n. In Appendix B we further explain the mathematical analysis behind the result in Equation 1.

Authenticity Requirement. The SvPDD protocol sends roughly the same amount of self and p2p-messages. A message of one class is indistinguishable from that of the other; therefore, an attacker that modifies a protocol message, modifies with probability $\frac{1}{2}$ a p2p-message and with probability $\frac{1}{2}$ a self-message.

Assume that the total number of messages that near-MitM attacker modifies is *less than* $7\delta n$. Let ξ denote the number of p2p-messages that he modifies.

Since each message that the attacker modifies has probability $\frac{1}{2}$ to be 'p2p', the expected value of ξ is $E[\xi] < \frac{7}{2}\delta n$. However, in order to provide a false response, the attacker must modify messages of more than $\frac{1}{2}n \geq 4\delta n$ p2p-transactions of a particular SvPDD-session (since $0 < \delta \leq \frac{1}{8}$).

Hoeffding's inequality allows to bound the probability that $\xi \geq 4\delta n$ (and therefore, bound the probability that $\xi > \frac{1}{2}n$); i.e., that ξ deviates from its expected value by at least $\frac{1}{2}\delta n$, see Equation 2:

$$\Pr[\xi \geq 4\delta n] \leq e^{-2(\frac{1}{2}\delta n)^2} = e^{-\frac{1}{2}(\delta n)^2} \tag{2}$$

Complementary, assume that the total number of messages that the near-MitM attacker modifies is *at least* $7\delta n$. Let ξ' denote the number of self-messages that he modifies. Since each message that the attacker modifies has probability $\frac{1}{2}$ to be 'self', the expected value of ξ' is $E[\xi'] \geq \frac{7}{2}\delta n$. However, in order to avoid a MitM alert, the attacker must modify less than $3\delta n$ self-messages.

Hoeffding's inequality allows to bound the probability that $\xi' < 3\delta n$; i.e., that ξ' deviates from its expected value by at least $\frac{1}{2}\delta n$, see Equation 3:

$$\Pr[\xi' < 3\delta n] \leq e^{-2(\frac{1}{2}\delta n)^2} = e^{-\frac{1}{2}(\delta n)^2} \tag{3}$$

The bounds in Equations 2 and 3 show that the probability that the attacker succeeds in violating the authenticity property is a negligible function in n. In Appendix B we further explain the mathematical analysis behind the results in Equations 2 and 3.

5 Plug-and-Play IP Security

This section presents Plug-and-Play IP Security (PnP-IPsec), a protocol that establishes an IPsec tunnel [15] between two network gateways without coordinated administration and without relaying on a public key infrastructure.

Figure 4 illustrates a typical deployment topology for PnP-IPsec. The protocol's goal is that if there are two communicating hosts, Alice and Bob, behind two PnP-IPsec gateways, then the gateways will *automatically* establish an IPsec tunnel to secure all communication between their networks. In this section we assume that there are no intermediate PnP-IPsec gateways (such as $\mathsf{GW_C}$ in Figure 4), Section 6 extends the protocol to handle this scenario.

PnP-IPsec builds on SvPDD; namely, each gateway uses SvPDD to retrieve and validate the IPsec configuration from its peer. Figure 5 illustrates the three phases that compose PnP-IPsec, which we describe in the following three subsections. In the fourth subsection we describe the protocol's security properties.

5.1 Initiation Phase

PnP-IPsec is initiated by a gateway, $\mathsf{GW_A}$, when it forwards a packet from Alice to Bob. This is the *trigger* packet illustrated in Figure 5. The initiation is

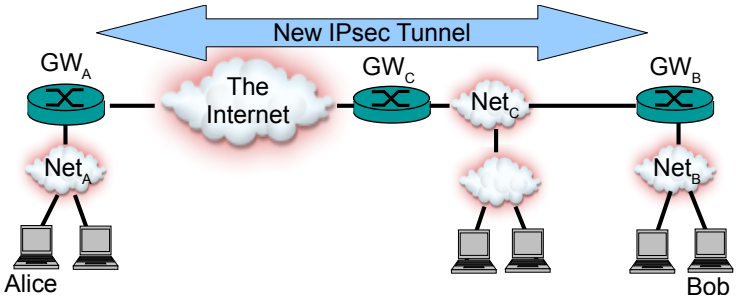


Fig. 4. PnP-IPsec Deployment Topology. Alice and Bob are communicating hosts; PnP-IPsec is deployed on $\mathsf{GW_A}$ and $\mathsf{GW_B}$ and establishes an IPsec tunnel between them.

probabilistic: a trigger packet initiates the handshake with a (configurable) probability $p > 0$; the lower p is, the lower PnP-IPsec overhead and the more time is required to establish a tunnel.

$\mathsf{GW_A}$ begins the PnP-IPsec handshake by initiating an SvPDD-session to retrieve the IPsec configuration of the gateway closest to Bob. The response configuration includes the following three elements, which in 'classic setup' of IPsec are manually configured by the network administrator at both gateways.

1. The gateway's (responder) IP address; which is the encapsulation end-point for tunneled traffic.
2. The gateway's public key; used to secure IPsec messages.
3. The network address block behind the gateway; traffic to this network block will be encapsulated.

Additionally, the response includes a client puzzle [3] and a cookie that allows the responder to re-generate the puzzle (without keeping state). The initiator solves this puzzle in order to request the responder to initiate a PnP-IPsec handshake in the opposite direction; as we describe in the last phase of the handshake. The use of a client-puzzle protects the responder from a denial of service (DoS) attack that persuades him to initiate PnP-IPsec handshakes with arbitrary peers (see security discussion in Subsection 5.4).

Since $\mathsf{GW_A}$ does not know the address of Bob's gateway, SvPDD-queries (i.e., IPsec configuration requests) are sent to Bob's address. The queries traverse the route from the anonymity network to Bob, allowing Bob's gateway, $\mathsf{GW_B}$, to intercept the queries and respond. See phase 1 in Figure 5.

The responder ($\mathsf{GW_B}$) only handles the queries if it is unaware of another PnP-IPsec gateway 'behind it' that is also a gateway of Bob. The reason is that PnP-IPsec should establish IPsec tunnels between the closest gateways to Alice and Bob (the communicating hosts) in order to protect their communication from intermediate malicious nodes (MitM attackers). In Section 6 we show how gateways automatically learn which of their subnets have a 'closer' gateway.

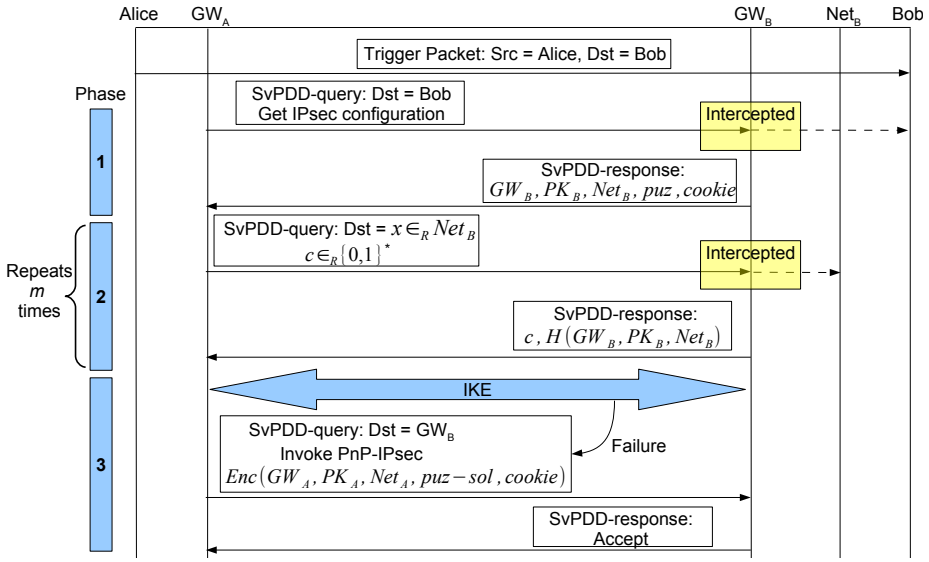


Fig. 5. PnP-IPsec Diagram. Dashed arrows mark destinations of intercepted packets.

5.2 Validation Phase

In this phase the initiator validates that the responder controls the claimed network address block (provided in the Initiation phase). This phase is similar to the network block validation process that we presented in [7], except that the messages here are sent over SvPDD in order to cope with a MitM attacker (see analysis in Section 5.4); we briefly present the network block validation protocol.

Network block validation is composed of m parallel SvPDD-sessions (m is a security parameter), where in each session the initiator ($\mathsf{GW_A}$) picks a *random address* in the responder's ($\mathsf{GW_B}$) claimed network block and sends a challenge to it (each session is associated with a different address). If $\mathsf{GW_B}$ is indeed the gateway of that address, then it can intercept the challenge and respond; see phase 2 in Figure 5. If *all* challenges receive correct responses, then $\mathsf{GW_B}$ is validated to control the network block that it claimed.

The following describes the challenge and response messages.

Challenge. The challenge is an SvPDD-query for a random string, denoted by c.

Response. The response is the tuple $< c, H(\mathsf{GW_B}, \mathsf{pk_B}, \mathsf{net_B}) >$, where c is an echo of the challenge and $< \mathsf{GW_B}, \mathsf{pk_B}, \mathsf{net_B} >$ is $\mathsf{GW_B}$'s IPsec configuration; H is a cryptographic hash function.

When $\mathsf{GW_A}$ receives the response (returned by SvPDD after a challenge-response session completes), it verifies that the value c is correct. $\mathsf{GW_A}$ also verifies that the hash value matches that of $\mathsf{GW_B}, \mathsf{pk_B}$ and $\mathsf{net_B}$ which were received in the Initiation phase, in order to ensure that the responder does not change.

5.3 Invocation Phase

In the last phase, $\mathsf{GW_A}$ invokes IKE [14] and attempts to bootstrap IPsec (phase 3 in Figure 5), using the remote configuration $< \mathsf{GW_B}, \mathsf{pk_B}, \mathsf{net_B} >$.

If $\mathsf{GW_B}$ has the corresponding configuration of $\mathsf{GW_A}$ ($< \mathsf{GW_A}, \mathsf{pk_A}, \mathsf{net_A} >$), then IKE will establish an IPsec tunnel between the two gateways[2]. Otherwise, IKE aborts; in this case, $\mathsf{GW_A}$ requests $\mathsf{GW_B}$ to initiate a PnP-IPsec handshake in the opposite direction (see Figure 5). The request is an SvPDD-query which specifies $\mathsf{GW_A}$'s public IPsec configuration configuration, $< \mathsf{GW_A}, \mathsf{pk_A}, \mathsf{net_A} >$, as well as the solution to the client puzzle (i.e., proof of work) and cookie that $\mathsf{GW_B}$ sent in the Initiation phase. This request is encrypted using $\mathsf{GW_B}$'s public key; therefore, it does not leak the identity of the initiator ($\mathsf{GW_A}$), which is required in order to anonymize queries and use SvPDD (see Section 3.1).

When $\mathsf{GW_B}$ receives this request, it re-generates the puzzle using the cookie and verifies the solution of the puzzle. If the solution is correct, then $\mathsf{GW_B}$ may accept the request, if it is interested in setting up a tunnel with $\mathsf{GW_A}$ (e.g., this may depend on the Initiator's network, $\mathsf{net_A}$); otherwise $\mathsf{GW_B}$ rejects the request. If $\mathsf{GW_B}$ accepts, then it continues to the handshake's Validation phase.

In the Invocation phase of this second handshake both gateways will have each other's configurations (IKE can bootstrap IPsec). However, if IKE initiation does not succeed (on the second time), then a MitM is assumed to block IKE (preventing establishment of IPsec), and the gateways block the (clear-text) traffic between their networks.

5.4 Security Discussion

In this subsection we motivate the security properties of PnP-IPsec.

Discovery: PnP-IPsec gateways of communicating hosts quickly detect each other.

Assume that Alice sends packets to Bob. For every such packet, the probability that $\mathsf{GW_A}$ (Alice's gateway) initiates the PnP-IPsec handshake is p; namely, the probability that the handshake *does not initiate* after k packets is $(1-p)^k$, i.e., exponentially decreasing (since $p > 0$). When $\mathsf{GW_A}$ completes the PnP-IPsec handshake, i.e., retrieves and validates $\mathsf{GW_B}$'s public IPsec configuration, it triggers the handshake in the opposite direction. Namely, only a few packets travel between Alice and Bob before the gateways discover each other.

Authentication: a PnP-IPsec gateway learns the IPsec configuration from the correct responder, rather than a MitM attacker.

This property of PnP-IPsec follows from the authenticity property of SvPDD, since the configuration is obtained over an SvPDD-session (in the Initiation phase).

[2] Since the gateways run PnP-IPsec without coordination, it is likely that $\mathsf{GW_B}$ had already received $\mathsf{GW_A}$'s public IPsec configuration.

Correctness: a gateway only learns a correct configuration from its peer.

The gateway learns the configuration from the correct peer (the authenticity property). It is left to show that this configuration is also correct; namely, that a malicious responder cannot persuade the initiator that it controls a false network block. We now motivate why such malicious responder will not pass the Validation phase, i.e., the responder will not be able to provide a correct response for at least one challenge; we refer to [7] for further details.

Assume that the responder controls net_1, but advertises $\mathsf{net}_2 \neq \mathsf{net}_1$; namely, $\frac{|\mathsf{net}_1 \cap \mathsf{net}_2|}{|\mathsf{net}_2|} = \alpha < 1$. The probability that the responder receives all challenges is α^{-m}, where m is the number of challenge-response sessions (and number of different challenge destination addresses); i.e., the probability that a gateway does not control the entire network block that it claims, but passes the Validation phase, is negligible in m. In practice the ratio is often $\alpha \ll 1$, because ISPs use CIDR address allocation; we refer to [7] for further analysis of the network block validation technique.

Resilience to DoS: PnP-IPsec does not open a new denial of service attack vector on the responder.

We show that: first, PnP-IPsec has low communication and computational requirements from the responder; and second, the responder does not keep any state during the handshake.

First, in terms of communication load, the responder only sends one message (response) for every message (query) that the initiator sends. In terms of computation, the responder generates a client puzzle in the Initiation phase, which is very efficient (client puzzles [3] are means to mitigate DoS attacks). An initiator can cause the responder to initiate a handshake, however this requires solving the responder's puzzle, which has significant computational overhead.

Second, in terms of memory, the responder does not keep state per-peer or between requests: (1) the responder provides its (single, global) public IPsec configuration during the Initiation phase; (2) the responder only requires the challenge-field specified in the challenge packet in order to generate the corresponding response during the Validation phase; (3) the responder re-generates, rather than saves, the client puzzle (using the cookie) when it receives a request to initiate a PnP-IPsec handshake in the Invocation phase.

6 Extending PnP-IPsec for Multiple Gateways

PnP-IPsec should establish an IPsec tunnel between the gateways that are 'closest' to the communicating hosts; these are $\mathsf{GW_A}$ and $\mathsf{GW_B}$ in the example network topology that is illustrated in Figure 4. However, an intermediate *non-malicious* gateway, such as $\mathsf{GW_C}$, who is unaware of the existence of a gateway behind it (i.e., $\mathsf{GW_B}$) may unintentionally 'hijack' the PnP-IPsec handshake by responding to the Initiation-phase message that $\mathsf{GW_A}$ sends to Bob (see Figure 5). This

section describes the discovery process for lower-tier gateways, where $\mathsf{GW_C}$ learns that $\mathsf{net_B}$ is, in-fact, under control of $\mathsf{GW_B}$[3].

6.1 Proactive Gateway Discovery

In order to detect higher-tier gateways, a PnP-IPsec gateway sends a discovery message to a random address outside of its network block. This message specifies a random identifier, the gateway's public key and its network block.

If a gateway, say $\mathsf{GW_B}$ (see Figure 4), connects to the Internet via another PnP-IPsec gateway, $\mathsf{GW_C}$, then $\mathsf{GW_C}$ will intercept the discovery message and initiate a network block validation process with $\mathsf{GW_B}$. Network block validation is similar to that described in Section 5.2 except that it does not run over SvPDD; i.e., the challenges and responses are transmitted directly to their destinations (and not via the anonymity network). The reason that we do not employ SvPDD is that, in this case, protection against MitM attackers is not required: if there is a MitM attacker between $\mathsf{GW_C}$ and $\mathsf{GW_B}$ who hijacks the PnP-IPsec handshake, then he will be detected since PnP-IPsec runs over SvPDD (our goal in this section is only to detect intermediate non-malicious PnP-IPsec gateways).

If the network block validation completes successfully, then $\mathsf{GW_C}$ learns that $\mathsf{net_B}$ is in-fact under control of $\mathsf{GW_B}$. In this case, $\mathsf{GW_C}$ will not respond to future PnP-IPsec messages sent to or from $\mathsf{net_B}$ (see network illustration in Figure 4), which will allow $\mathsf{GW_A}$ and $\mathsf{GW_B}$ to use PnP-IPsec and establish a tunnel.

Dynamic Network Topologies. New PnP-IPsec gateways can unexpectedly set-up while others can suddenly shut-down. Therefore, PnP-IPsec gateways periodically send discovery messages, in order to allow new higher-tier gateways to detect their presence (and network block ownership).

Additionally, gateways (such as $\mathsf{GW_C}$ in Figure 4) periodically send challenges to their subnets (such as $\mathsf{net_B}$) that are marked as controlled by lower-tier gateways (i.e., $\mathsf{GW_B}$) in order to ensure that the lower-tier gateways are still available and control their subnets.

Finally, when a PnP-IPsec gateway (such as $\mathsf{GW_B}$) gracefully shuts down, it sends a prune message to its higher-tier gateway ($\mathsf{GW_C}$) in order to revoke ownership over the subnet ($\mathsf{net_B}$) immediately.

7 Implementation and Deployment

We implemented PnP-IPsec as well as the underlying SvPDD protocol, as an open-source application for Linux gateways; our implementation is available at `http://pnpipsec.sourceforge.net/`.

[3] A malicious $\mathsf{GW_C}$ may not follow the protocol described in this section and hijack connections to $\mathsf{net_B}$, in this case $\mathsf{GW_B}$ will identify $\mathsf{GW_C}$ as a near MitM (since PnP-IPsec builds over SvPDD).

In order to deploy PnP-IPsec, the network administrator only needs to install our application on the local gateway and provide it with the gateway's private/public key pair (since the keys are not signed, e.g., by a CA, they may also be automatically generated at install time). PnP-IPsec learns the reminder of the local IPsec configuration, i.e., gateway's IP address and the network address block behind it, by reading the routing table. The configuration also includes the near-MitM threshold (δ), the probability to initiate a PnP-IPsec handshake (p), and security parameters (n, m), which have default values that may be modified by the administrator. The following is an example of a deployment command:

```
PnpIPsec.py private-key-file public-key-file
```

In terms of efficiency, our implementation establishes an IPsec tunnel between two gateways, whose networks communicate at the rate 1mbps, in approximately two minutes; each gateway sends less than 3MB of PnP-IPsec traffic. This measurement is by using the default parameters: $\delta = \frac{1}{10}, p = \frac{1}{100}, n = 40, m = 20$.

8 Conclusions and Future Work

Our main conclusion from this work is that while 'conservative' key infrastructures such as key distribution centers and certification authorities may be inconvenient for deployment of some protocols, other infrastructures may be suitable. In particular, we showed how available anonymity networks can be utilized to allow convenient and secure deployment of IPsec.

We presented SvPDD, a query-response protocol that utilizes an anonymity infrastructure to cope with the man-in-the-middle threat model. We built PnP-IPsec over SvPDD, which allows automatic establishment of IPsec tunnels. We provided an open-source implementation of PnP-IPsec and hope that this work will increase the deployment of the IPsec defense.

Future Work. The model considered in this paper, of using an available anonymity infrastructure in order to authenticate public keys and data, is practical. It is therefore desirable to formally define this model which may benefit other scenarios and protocols.

Furthermore, we believe that our protocols could further be improved. In terms of efficiency, the use of anonymity networks to relay messages usually comes at the price of encapsulation overhead. Can we improve the performance of SvPDD without jeopardizing its security requirements? In terms of functionality, can we extend PnP-IPsec to support setup of multicast IPsec tunnels?

9 Acknowledgements

We would like to thank Adrian Perrig and the anonymous referees for their helpful comments and suggestions. This research was supported by grant 1354/11 from the Israeli Science Foundation (ISF), and a grant from the Ministry of Science and Technology, Israel.

References

1. Raz Abramov and Amir Herzberg. TCP Ack Storm DoS Attacks. *Computers & Security*, 33:12–27, 2013.
2. Mansoor Alicherry and Angelos D. Keromytis. DoubleCheck: Multi-Path Verification against Man-in-the-Middle Attacks. In *ISCC*, pages 557–563. IEEE, 2009.
3. Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. DoS-Resistant Authentication with Client Puzzles. In Bruce Christianson, Bruno Crispo, and Michael Roe, editors, *Security Protocols Workshop*, volume 2133 of *Lecture Notes in Computer Science*, pages 170–177. Springer, 2000.
4. Comodo^{TM}. Incident Report. Published online, `http://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html`, March 2011.
5. Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*, pages 303–320. USENIX, 2004.
6. W. Eddy. TCP SYN Flooding Attacks and Common Mitigations. RFC 4987 (Informational), August 2007.
7. Yossi Gilad and Amir Herzberg. LOT: A Defense Against IP Spoofing and Flooding Attacks. *ACM Transactions on Information and System Security*, 15(2):6:1–6:30, July 2012.
8. Yossi Gilad and Amir Herzberg. Plug-and-Play IP Security: Anonymity Infrastructure Instead of PKI. In *ESORICS*, LNCS, pages x–y. Springer, 2013. To be published.
9. John Gilmore. FreeS/WAN. Published online, `www.freeswan.org`.
10. Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.
11. Amir Herzberg and Haya Shulman. Stealth DoS Attacks on Secure Channels. In *Proceedings of Network and Distributed Systems Security (NDSS)*. Internet Society, Feb. 2010.
12. R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. RFC 2459 (Proposed Standard), January 1999. Obsoleted by RFC 3280.
13. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography from Anonymity. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 239–248, 2006.
14. C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996 (Proposed Standard), September 2010. Updated by RFC 5998.
15. S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005. Updated by RFC 6040.
16. Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew K. Wright. Timing Attacks in Low-Latency Mix-Based Systems. In Ari Juels, editor, *Proc. Financial Cryptography*, pages 251–265. Springer-Verlag, LNCS 3110, Feb. 2004.
17. Moxie Marlinspike. Convergence. Published online, `http://convergence.io`, 2011.
18. The Tor Project. Tor Metrics Portal. Published online, `https://metrics.torproject.org/graphs.html`, April 2013.
19. M. Richardson. A Method for Storing IPsec Keying Material in DNS. RFC 4025 (Proposed Standard), March 2005.
20. M. Richardson and D.H. Redelmeier. Opportunistic Encryption using the Internet Key Exchange (IKE). RFC 4322 (Informational), December 2005.

21. K. Sampigethaya and R. Poovendran. A Survey on Mix Networks and Their Secure Applications. *Proceedings of the IEEE*, 94(12):2142–2181, 2006.
22. Claudia Schmeing. FreeS/WAN Announcement. Published online, `http://www.freeswan.org/ending_letter.html`, 2004.
23. Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2009.
24. J. Touch, D. Black, and Y. Wang. Problem and Applicability Statement for Better-Than-Nothing Security (BTNS). RFC 5387 (Informational), November 2008.
25. Dan Wendlandt, David G. Andersen, and Adrian Perrig. Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing. In Rebecca Isaacs and Yuanyuan Zhou, editors, *USENIX Annual Technical Conference*, pages 321–334. USENIX Association, 2008.
26. N. Williams and M. Richardson. Better-Than-Nothing Security: An Unauthenticated Mode of IPsec. RFC 5386 (Proposed Standard), November 2008.
27. T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), January 2006.

A SvPDD Construction over Tor

In this section we provide the details of our construction of SvPDD over the anonymity network Tor [5]. We briefly present the Tor network, discuss the challenges in instantiating SvPDD over it and provide mitigations to those challenges.

A.1 The Tor Anonymity Network

When a source (Tor-client) wishes to communicate with a destination over Tor, it first retrieves a list of available proxies from centralized directory servers and establishes a 'circuit' of few proxies. When a source sends a message, it encapsulates the message in cryptographic layers (using keys of the proxies) and sends it to the first ('entrance') proxy in the circuit, which removes the top layer of encapsulation and sends the payload to the next proxy who performs the same procedure; this scheme is called 'onion routing'. The last proxy, called the 'exit proxy', receives the message and sends it to the destination.

The destination responds to the exit proxy who encapsulates the message using a symmetric key that is shared with the source, and sends the encapsulated message to the previous proxy in the circuit. The message travels back from one proxy in the circuit to the other, until it reaches the client who can remove the encapsulation layer.

In Tor, the exit proxy knows the destination, but not the source; in contrast, the entrance proxy knows the source, but not the destination. Intermediate proxies only know their neighboring proxies in the circuit.

A.2 Challenges and Attack Vectors

Tor was designed to provide anonymity in environments where a MitM attacker is present. However, there are still some known attack vectors on Tor anonymity that would allow such attackers to differentiate between self and p2p-messages. We briefly describe the known attack vectors relevant to implementation of SvPDD over Tor and in the following subsection present mitigations.

Malicious Directory Servers. An attacker that controls a majority of the directory servers is able to provide Tor clients a malicious proxy list, that is composed entirely of proxies under attacker's control; thereby forcing clients traffic to be routed only via the malicious proxies.

This attack breaks Tor's anonymity guarantees for all its clients; therefore, Tor designers made efforts to mitigate it: first, there are 10 directory servers, a client only learns of proxies that are considered available by a majority of those servers; an attacker needs to corrupt at least 5 such servers in order to modify the proxy list that a client receives. Second, the directory servers are located in geographically disperse locations and are constantly monitored. We therefore believe that this attack vector is unlikely.

Time and Delay Side Channel. An attacker may correlate a message's transmission time to other messages in order to identify its type. Furthermore, a powerful attacker can infect the Tor network with malicious proxies by joining infected hosts to the network (proxies are volunteer hosts). If the attacker controls the *entrance and exit* proxies in the circuit, then he can identify the querier and responder by correlating messages as they enter and exit the Tor network according with the delay inside the network [16].

Malicious Exit-Proxies. The exit proxy receives plain-text messages to/from the destination; a malicious proxy can modify these messages. This proxy has, in fact, MitM capabilities, but only for communication in the specific circuit; it cannot modify communication that uses a different exit node.

A.3 Mitigations

In this subsection we show how we can construct SvPDD over Tor and mitigate the attack vectors described above.

Time and Delay Side Channel - Mitigation. SvPDD messages are short and do not require high quality of service (such as low latency or a high-speed connection). Therefore, we can choose the proxies in a circuit uniformly from the list of available proxies (provided by the directory servers); this is in contrast to the standard Tor client which chooses proxies in the circuit according to stability, latency and available bandwidth.

Furthermore, since SvPDD does not require a constant connection between the peers, our implementation uses different circuits in each transaction; mitigating the risk that an adversary repeatedly controls both the entrance and exit proxies.

Finally, we introduce a randomized jitter to SvPDD communication. First, we introduce a short randomized bounded delay before sending each message. Second, we randomize the number of proxies that participate in each circuit.

Malicious Exit-Proxies - Mitigation. SvPDD must avoid false MitM detection in case of a malicious exit proxy who modifies traffic. This is obtained by concluding that a MitM was detected only after several self-transactions fail (see Section 3.2). The probability of using malicious exit nodes for a sequence of transactions drops exponentially in the security parameter n (number of transactions in each session).

B Further Details of SvPDD Security Analysis

In Section 4 we used Hoeffding's inequality to bound the probability that a MitM receives enough messages to force a false result of an SvPDD session. This appendix provides the details of our mathematical analysis. We focus on the 'no false alert' requirement and computation of Equation 1. The details behind the authenticity requirement and computation of Equations 2 and 3 are similar.

Let the random variable $t_i = \{0, 1\}$ be 0 if the attacker does not obtain a message (query or response) that belongs to transaction i, and 1 otherwise ($1 \leq i \leq n$). Since the attacker is a far MitM with respect to both peers, he receives every message between them with probability δ. Hence, the probability that he receives at least one of the two messages in a particular transaction is no more than 2δ ($\forall i : \Pr[t_i = 1] \leq 2\delta$). It follows that the expectancy of the number of messages that the attacker receives, denoted by η, is no more than $E[\eta] = \sum_i \Pr[m_i = 1] \leq 2\delta n$. However, in order to cause a false alert, the attacker must modify at least $3\delta n$ messages (see Section 3.2).

Hoeffding's inequality allows to bound the probability that a sum of random variables (η) deviates from its expectancy ($E[\eta]$) at least by a threshold t, as shown in Equation 4. In our case, $t = 3\delta n - E[\eta] \geq \delta n$, which provides the result of Equation 1.

$$\Pr[\eta - E[\eta] \geq t] \leq \Pr[\eta - E[\eta] \geq \delta n] \leq e^{-2t^2} \leq e^{-2(\delta n)^2} \tag{4}$$