Seif Protocol Version 0

Aashish Sheshadri, Rohit Harchandani and Douglas Crockford

April 11, 2017

Abstract

We describe a communication protocol that establishes secure sessions between parties in a single round trip. The Seif protocol is part of the overarching project, Seif[1], an open initiative tasked to make the web secure and application friendly. It is a messaging protocol that is session based, thus leveraging capabilities of session architectures.

1 Introduction

Today HTTP[4] is the most commonly used protocol for moving resources, delivering applications, and communicating over the Internet. While HTTP builds on top of the TCP protocol, which is connection oriented, HTTP presents a Request-Response model that is not the best choice for modern applications[8]. It is also highly tolerant of errors that compromise security. HTTP supports negotiation of many aspects of the data being communicated, which adds considerable complication but is seldom required by today's uses of the protocol.

SSL/TLS[9] is a cryptographic protocol for secure network communications. Today, it is the standard means of securing web traffic carried over HTTP. However, SSL has a number of deep and fundamental problems. Its unnecessarily complex nature has resulted in bugs being commonplace in many of its most widely used implementations, as well as the repeated discovery of significant flaws in the protocol's design itself[7, 3]. In the protocol's 20+ year history it has accreted further complexity but not stability. Its dependence on third party certificate authorities introduces large scale, structural security vulnerabilities. It makes poor use of the network, requiring a minimum of three round trips to set up a secure connection.

There have been several attempts made to make connections persistent, more secure, or less complex. However, proposed solutions tend to build on top of the existing HTTP or SSL specifications by having the end user experience be the same as that provided by HTTP[1]. While we acknowledge this as a direction that has value, our strong belief is that today's security goals necessitates a ground-up change.

In this paper we will present an application layer protocol with a strong security guarantee as an alternative to HTTP over SSL/TLS and similar protocols.

[1] www.seif.place

1.1 Goals and Features

The Seif protocol is designed to keep things simple, enabling ease of reasoning about reliability, security, and correctness. The protocol is motivated by the following goals:

- Enable secure connections, between two parties while avoiding the need for third party entities.
- Enable persistent and highly reliable connections.
- Reduce connection times by reducing the number of round trips.
- Strict error policy.
- No negotiation on security parameters.

The Seif protocol is unique in its design in the following ways:

- It is a session-oriented, message based protocol, rather than being based the request/response paradigm embodied by HTTP; a feature most sought after by modern applications.
- The protocol does not allow negotiation (with the exception of Seif version information). This protects the session against renegotiation attacks and protocol downgrade attacks.
- In the case where Alice is trying to connect to Bob, prior to initiating the connection, the protocol assumes Alice has access to Bob's public key. The trust management system which helps accomplish this is will not be described and is out of the scope of the protocol definition.
- As a result of the above assumption, there is no dependence on certificate authorities or any other third parties during the Seif handshake (described in section 2.6). This not only provides protection against attacks on the certificate authorities, but also helps improve performance.
- The handshakes are thus greatly simplified and secure connections can be set up in just one round trip.
- It provides forward secrecy as long as the private keys of both the parties are kept private.
- It relies on secure symmetric link encryption, public key encryption, and random number services to achieve security strength equivalent to 256 bits.

2 Protocol

We will now describe the protocol in detail. Sections will cover establishing connections, sending messages, redirects, and error handling.

2.1 Terms and Specifications

For Seif version 0, we use the following standards and algorithms. Future versions may substitute different algorithms offering a higher security guarantee. However, nothing in the fundamental logic of the Seif connection initiation handshake is dependent on the particular choice of algorithms given here.

- Public-key Encryption using ECC-521[5, 6].
- Symmetric Encryption using AES-256[2].
- Hashing using SHA3-256.

The protocol recommends that the secure session be established on top of a highly reliable, persistent network connection with ordered and error-checked delivery of the data stream. In this paper, we are assuming that the underlying network connection is established using TCP, but in practice, any protocol with the features just listed can be used.

The protocol also assumes that any implementation has access to a high quality source of random numbers.

2.2 Link Encryption

Seif uses AES-256 in GCM mode to encrypt the communication stream. Since the channel will look exactly the same for an identical stream, the protocol modulates the channel with a random stream of numbers. The random stream is generated using Xorshift+ Random Number Generator (RNG). The modulation scheme used is simple XOR of individual packets. This modified process is illustrated in Figure 1.

The protocol requires the RNG be seeded differently for incoming and outgoing packets. Since the session key is available to both the sender and the receiver and is sufficiently random, Seif uses this to seed the RNGs. The sender uses the first half of the key to seed its RNG for outgoing packets and expects the incoming packets to be modulated with a random stream from a RNG seeded using the second half of the key. The case is the mirror opposite at the receiver end.

2.3 CSPRNG: Cryptographically Secure Random Numbers

A secure and reliable RNG is essential to enabling a strong cryptographic backbone. The protocol thus requires such a generator be securely accessible. The protocol uses random numbers to realize encryption keys, session secrets, and generating public-private key pairs.

The RNG is required to posses all properties of a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG). Furthermore, the protocol recommends that the generator be capable of managing a large internal state and have an astronomically long period.

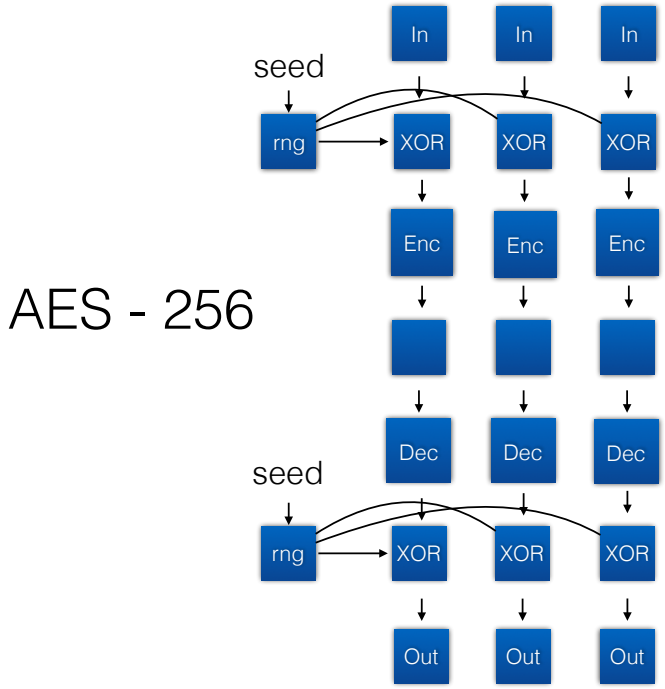

Figure 1: Describes modified AES-256 algorithm used by Seif. 'In' packets are plain text packets. A RNG is seeded with a seed 'a'. 16 Byte blocks from the RNG are XOR'd with the plain text blocks of the same size. The resulting XOR'd blocks are encrypted using AES-256 resulting in 'ENC' blocks. These blocks are then transmitted over the wire. The receiver decrypts the blocks using AES-256 to 'DEC' blocks. The same RNG is used at the receiver and is seeded with seed 'a'. Using part of the session key to seed the RNG is recommended since both the sender and receiver have access to the same key. Random 16 byte blocks from the RNG are XOR'd with 'DEC' blocks to recover the plain text sent over the wire as 'Out' packets.

2.4 Seif Party

The Seif protocol enables communication between Seif parties. A Seif party is the representation of the real world application entity at the Seif protocol layer. This is similar to a "socket", which is the representation of the application entity at the TCP layer. Thus a party can represent a server listening for connections, accepting connections, and sending/receiving messages over the Seif connection. On the other hand, it can also represent a client application initiating connections with the server party and sending/receiving messages over the Seif connection.

A party has the following capabilities:

- Connect to other Seif parties.
- Send messages over the Seif connection.

- End the Seif connection.
- Listen for Seif connections.
- Handle unexpected error events.
- Redirect sessions to other Seif parties.

2.4.1 Creation

Each party has an ECC-521 (Elliptic curve cryptography) {publickey, privatekey} pair, making this party uniquely identifiable in the world.

2.4.2 Initialization

The party can be initialized in several ways (which are not governed by the protocol and hence out of the scope of this paper) to get access to its {publickey, privatekey} pair and an CSPRNG.

2.5 Seif Blobs

All data on the wire is sent as binary buffers. Data is serialized as JSON before it is converted to a binary buffer. Identification and naming is achieved in the following way.

- A two byte block sent is the clear encodes length of a following encrypted/plain identifier JSON.
- The identifier JSON encodes "`type`" of the record, i.e. handshake or message, as will be discussed in the following sections.
- The identifier JSON also includes a field, "`blobs`", which is an array of blob IDs.
- Each blob ID is a JSON with "`id`" encoding name of the blob, "`length`" encoding length of the blob and optionally other fields as will be described later.
- Following the identifier JSON are binary buffers characterized by "`blobs`" described in the identifier JSON.

2.6 Seif Handshake

Consider the case where Alice is trying to connect to Bob for the first time ever. Lets assume that a TCP connection has been established between the two parties. The Seif handshake process is illustrated in Figure 3. Now we will describe the JSON message specifications to setup a valid Seif session.

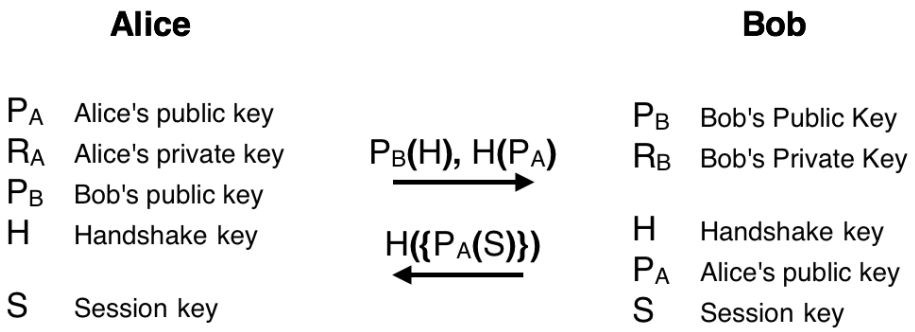

Figure 2: Describes a simplified Seif handshake flow detailed in section 2.6.

2.6.1 Hello

Alice generates a new, single-use symmetric key, the "`handshakeKey`". Alice sends Bob the `Hello` message, containing:

- "`version`": The Seif protocol version number. For the protocol described in this document, the version number is 0.
- "`handshakeKey`": The handshake key, encrypted using Bob's public key.
- "`helloData`": JSON containing Alice's public key and other optional fields. The "`helloData`" JSON is encrypted using the handshake key.
- "`connectionInfo`": JSON containing optional context for the connection. This JSON is not encrypted. Possible use cases include routing the connection or to provide context on a redirect.

The encrypted "`handshakeKey`" and "`helloData`" are transmitted as binary buffers as part of the `Hello` Seif blob.

2.6.2 AuthHello

Bob receives the `Hello` message and is able to decrypt the handshake key using his own private key and in turn gets access to Alice's public key. Bob generates a new, symmetric key, the "`sessionKey`", then sends Alice the `AuthHello` message to authenticate Alice and begin the session. This message is encrypted using the handshake key. The `AuthHello` message contains the following:

- "`sessionKey`": The session key, encrypted using Alice's public key.

The encrypted "`sessionKey`" is transmitted as a binary buffer as part of the `AuthHello` Seif blob.

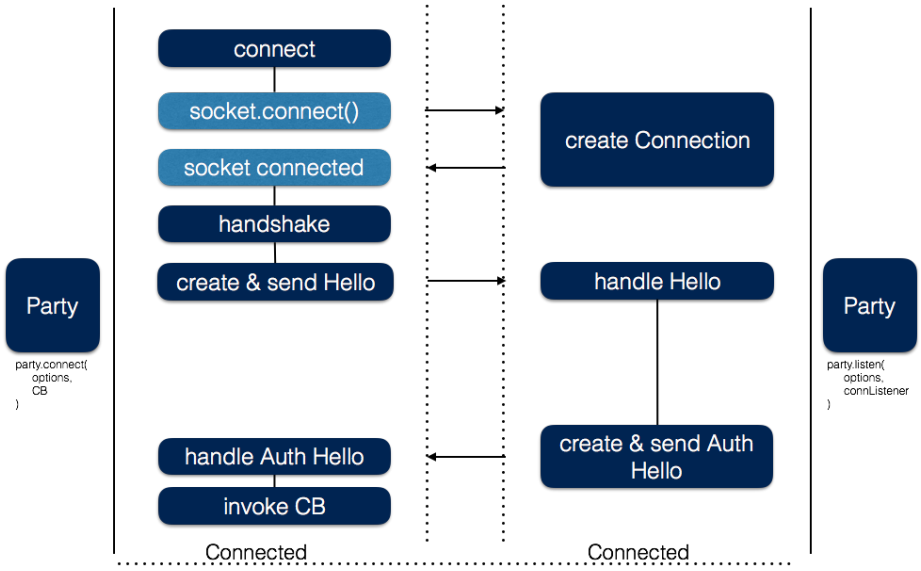

Figure 3: Describes the Seif handshake flow as described in section 2.6. The party on the left is Alice who is trying to connect to Bob, the party on the right. The "`handshake`" begins once the TCP connection is established. Alice sends the first "`Hello`" message as described in section 2.6.1. Bob processes it and sends the "`AuthHello`" message as described in section 2.6.2. At this point the Seif handshake is complete and Alice and Bob can send messages to each other over the established secure connection.

2.6.3 Session proceeds

Alice receives the `AuthHello` message, and is able to use the handshake key (which she generated) and her own private key to obtain the session key. Bob and Alice now both know the session key, which they use to encrypt all message traffic between them for the duration of the session.

2.7 Redirects

Redirects in Seif are applicable to sessions that have already been established. A session can be redirected to another listening Seif party for a variety of reasons. To describe this process and the types of redirects, lets have Bob introduce Alice to Carol. Alice is connected/connecting to Bob. Bob decides to redirect the session to Carol.

- To initiate the redirect, Bob sends a `Redirect` message to Alice. This message includes the following:
  - "`address`": Carol's address.
  - "`publicKey`": Carol's public key.
  - "`permanent`": Indicating if this is a permanent redirect.
  - "`redirectContext`": Optional context to the redirect.

- Alice, on receiving the `Redirect` message, ends her session with Bob
- Next, Alice initiates a connection to Carol.
- Alice and Carol can now establish their own session over this new connection, using the Seif handshake as described in section 2.6 above, without Bob having any knowledge of their session parameters.
- In the case where the optional `redirectContext` was included in the redirect, it will be sent as `connectionInfo` as part of the initiating `hello` to Carol.
- In the case of a permanent redirect, Alice is expected to connect to Carol instead of Bob on future connections.

2.8 Messages

The parties communicate by sending and receiving messages. A party can send a JSON object, a buffer or an array of JSON and buffers. All messages are encoded as Seif blobs as described in the Section 2.5 with an additional "`type`" field added to the blob ID. This field encodes if the message is a JSON or a buffer.

Seif supports two types of messages, `Send` (Section 2.8.1) and `Status Send` (Section 2.8.2). These messages are differentiated by guarantees on delivery and receive confirmation.

2.8.1 Send

A message of type '`Send`' will be acknowledged upon delivery to the intended party. In the event of network failure or connection breakdown the initiating party is notified of all messages that failed to be delivered.

2.8.2 Status Send

'`Status Send`' is the most basic of message types. This message is not expected to be consistently delivered. Specifically, the protocol need not notify the sender on delivery, nor is the message guaranteed to be delivered. This is designed for use-cases such as telemetry, logging and games.

2.9 Error Handling

Seif adopts a very simple strategy for error handling. For most errors, such as encryption failures, bad JSON record failures, etc. the Seif party closes the underlying TCP connection. The protocol attempts error recovery only in the event of a protocol version mismatch.

3 Security Analysis

- Seif achieves forward secrecy when making a connection between two parties. This is because an attacker requires both the parties private keys to break security guarantees.

- Seif's modified link encryption procedure enables a random looking stream on the wire even if the packets being transmitted are the same. For instance, a stream of acknowledgment packets will look like random packets on the wire. This makes it difficult for an observer to associate meaning to the packets thus dissuading attempts to replay packets.
- The error handling strategy significantly reduces error messages being sent over the network giving attackers little opportunity to discern causality. Although this might be inconvenient for protocol implementations, the security benefits make it a reasonable trade-off.
- In the case when Alice is connecting to Bob, given that Alice is confident of possessing Bob's public-key, Seif will establish a secure connection between the two parties voiding third-party involvement of certificate authorities.
- The protocol ensures security by minimizing the need for negotiations.

References

[1] Mike Belshe and Roberto Peon. Spdy protocol—draft 3.1. *URL http://www. chromium. org/spdy/spdyprotocol/spdy-protocol-draft3-1*.

[2] Pete Chown. Advanced encryption standard (aes) ciphersuites for transport layer security (tls). Technical report, 2002.

[3] Wassim El-Hajj. The most recent ssl security attacks: origins, implementation, evaluation, and suggested countermeasures. *Security and Communication Networks*, 5(1):113–124, 2012.

[4] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1. Technical report, 1999.

[5] Neal Koblitz, Alfred Menezes, and Scott Vanstone. The state of elliptic curve cryptography. In *Towards a quarter-century of public key cryptography*, pages 103–123. Springer, 2000.

[6] Kristin Lauter. The advantages of elliptic curve cryptography for wireless security. *IEEE Wireless communications*, 11(1):62–67, 2004.

[7] Christopher Meyer and Jörg Schwenk. Lessons learned from previous ssl/tls attacks-a brief chronology of attacks and weaknesses. *IACR Cryptology ePrint Archive*, 2013:49, 2013.

[8] Lucian Popa, Ali Ghodsi, and Ion Stoica. Http as the narrow waist of the future internet. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 6. ACM, 2010.

[9] Eric Rescorla. *SSL and TLS: designing and building secure systems*, volume 1. Addison-Wesley Reading, 2001.