Building scalable and secure L2 and L3 overlays with Host Identity Protocol

Dmitriy Kuptsov

2025 What if

Contents

1. Introduction

Back in end of *1960's* when the Internet was a rather small network, which was interconnecting major universities, governmental and military organizations, very little attention was devoted to security. Nowadays, when the Internet has become extremely sophisticated in structure, connecting billions of devices ranging from small IoT type devices to humongous data-centers, security has gained number one priority. In present days, a typical Intranet of an organization can include number of geographically separated branch-office networks (for example, consider a factory that has many SCADA devices and a mission control center that is miles and miles away). Since these networks geographically separated, connecting them becomes a necessity, and so is the security of these networks. This is when the layer-3 virtual private networks (*L3-VPN*) and layer-2 virtual private LAN services (*L2-VPLS*) solutions become handy. There are, however, other requirements that need to be taken into account. Scalability, resilience to various attacks, from man-in-the-middle to integrity violation attacks, to rather fundamental attacks on asymmetric algorithms (such as RSA, DSA and their elliptic curve counterparts, Diffie-Hellman and Eliptic Curve DH, for sampling) using, for example, *Shor's quantum computer algorithm* to factorize large numbers, and massive brute force attacks on hash algorithms should be considered thoroughly. With this in mind, in this work we present different security solutions, which can be used to build secure L2 and L3 overlay networks. We present the limitations of each solution and identify how they can be avoided using various VPLS and L3-VPN. With this in mind, we start with a background material on cryptography. Here we discuss various symmetric and asymmetric encryption algorithms, present the definition of hash functions, which considered secure nowadays, and discuss several key agreement algorithms. To make the discussion complete we present

the threat that quantum computers pose for such algorithms as RSA and DH, and discuss how post-quantum algorithms such as those that are based on lattice can be used as alternative to classical algorithms for encryption and signature constructions. Although, not considered as part of the present work, future work can be include the performance comparison of standardized RSA and DSA algorithms with the performance of lattice-based algorithms incorporated into for example Host Identity Protocol or event Transport Layer Security protocol. We than move on to discussion of TLS, SSL, IPsec, HIP and SSH protocols and how those can be used to achieve integrity and confidentiality of data transmitted over insecure channels. Afterwards, we move on to the discussion of the results we have obtained over the course of several years. Here, we discuss our practical experience with scalable Host Identity Protocol based L3-VPN and VPLS network which is build using the same protocol. We devote a separate section on hardware-accelerated versions of AES and SHA-256 algorithms. We conclude the results section with the analysis of the limitations of each solution and present the results for the various micro-benchmarking settings.

1.1 Questions

In this work, we ask several questions. These are not research questions, but rather practical questions we try to answer to ourselves in order to understand the usability of Python based security solution. Since our work focuses on the application of Host Identity Protocol (HIP) in VPN and VPLS settings we ask the following questions:

First, *what is the performance of the pure Python-based implementation of symmetric key encryption and decryption routines as well as hash methods and how do they compare to implementation, which uses special CPU AES and SHA-256 instructions.* Here our focus is on the microbanchmarking of two implementations of AES and SHA-256 hashing algorithm, identification of the bottlenecks and further recommendations for our prototype implementation of Host Identity Based VPLS and L3-VPN.

Second, *what is the scalability of Host Identity Protocol based VPLS and how does it perform in emulated environments such as Mininet.* Here we seek the answer to the question whether the HIP-VPLS is usable in environments close to real-life setups.

Third, *what is the performance of Python based HIP-VPLS on real hard-*

ware. By doing so we want to find the application niche of our security solution. In addition, we discuss the practical configuration of HIP-VPLS using central controller.

The next question *we want to answer relates to the deployment of scalable L3-VPN based on Host Identity Protocol*. Here we focus on rather different approach of building secure networks: We consider L3-VPN where nodes in different branches offices form separate broadcast domains, but still can communicate with each other (with assistance of IPv4 routing protocol). Here, we want to answer how to tackle the scalability issues of VPN network by adding hierarchy into the architecture?

2. Background

Since we are going to discuss the security protocols in this work, we begin this section with the shallow dive into cryptography basics. Here, we discuss symmetric and asymmetric cryptography algorithms, to make the description a little bit complete we show how RSA algorithm works, discuss the math behind Diffie-Hellman (DH) and its Elliptic Curve counterpart. We will also discuss the Shor's algorithm and its quantum computer implementation that theoretically can efficiently factorize big number. This algorithm, if powerful enough quantum computers will exist in the near future, puts the RSA algorithm - the major building block of modern security solutions - at risk of being cracked (once the modulus of RSA algorithm factorized into prime components, the private key of RSA algorithm can be easily recovered). We will conclude this part of the background material with the discussion of post-quantum computer public key encryption solution based on lattice (more specifically we will discuss Learning With Errors (LWE) algorithm). We believe that, eventually, this type of cryptography will be the replacement for traditional RSA and DH algorithms, which rely on the hardness of factorization of the big numbers and discrete logarithms. In the epilog of this section, we will put few words on how lattice public key cryptography can be used, for example, together with Host Identity Protocol.

In the second part of the background material, we will review the basics of the Host Identity Protocol, Transport Layer Security Protocol and Secure Shell Protocol, since these protocols are the main building blocks of secure tunneling protocols that we discuss in this work.

We will finalize the discussion of the background material with a short overview of various L2, L3 and L4 tunneling solutions, including L2 802.1Q QinQ tunneling, L3 Multi-Protocol Label Switching (MPLS) VPLS, L4 TLS and SSH tunneling.

2.1 Cryptography basics

Cryptography comes in many flavors: symmetric key cryptography (3DES, AES, Twofish, RC4) which, in turn, can be categorized into block cipher and stream cipher and asymmetric key cryptography (such as RSA, DSA, ECDSA). There are also key exchange protocols such as Diffie-Hellamn and Elyptic Cryptography DH for negotiation of common keys over insecure channels. Different algorithms applicable in different settings depending on requirements. Typically, as we will discuss later, symmetric key cryptography is used to protect data-plane traffic in networks, whereas, asymmetric-key cryptography is more applicable to the common key negotiation, authentication and identification purposes [**?**].

2.1.1 Symmetric cryptography

We start with the symmetric key cryptography. Common key and rather trivial operations such as permutations and substitutions are at the heart of any symmetric key cryptography algorithm. Although, this type of cryptography is efficient because of the usage efficient operations, it comes with a limitation though. In symmetric key cryptography, both sender and receiver need to share the same key, which complicates such important aspects as key distribution and revocation and so alone this encryption solutions a very hard to use alone in modern cryptosystems. Typically, asymmetric key cryptography such as RSA used to derive session keys – TLS, HIP and many other protocols follow this design idea.

Symmetric key cryptography comes in two different flavors: block and stream. For example, block cipher (such as AES, 3DES, Twofish [**?**]) use blocks of data (typically, the size of the block is 128, 160, 256 bits [**?**]), and encrypts or decrypts one block at a time. There are different modes of operation, though, for block ciphers, examples are counter mode and cipher block chaning. The latter one uses so-called initialization vector to add extra randomness into encryption process, and encryption of proceeding blocks depends on the output of the previous block. Modes of operations are important for security reasons. However, not all modes of operations useful and secure. For example, Electronic Code Book (ECB), while allow achieving fast processing and parallelization, is considered insecure in many settings. In Figure[] we demonstrate AES encryption in CBC mode:

The other type of symmetric key algorithms is stream cipher. Here the

encryption and decryption performed on separate bits, one bit at a time. CR4 is an example of stream cipher. Stream ciphers are extremely important in real-time processing, for example, Wi-Fi uses stream ciphers to encrypt the data plane traffic.

2.1.2 Asymmetric cryptography

Asymmetric key cryptography, in its simplest form, is the brilliant in the age of computing. Guessing from the name that this type of cryptography uses different keys for encryption and decryption does not require deep though. This property makes this group of algorithms suitable for various key distribution, revocation and signature ideas.

There is a magnitude of different asymmetric key security algorithms. RSA, DSA and its Elliptic curve variant ECDSA are the pillars of modern security solutions. But the flexibility of these schemes comes at an extra price of CPU cycles. All this makes these solutions inapplicable for securing data plane traffic, but only rather to secure control plane. In what follows, just to underpin the beauty of the math behind asymmetric key cryptography, we provide a description of RSA algorithm.

In RSA cryptosystem, the sender generates a pair of keys as follows: First, the sender chooses large enough two prime numbers p and q. Next, the sender computes $n = pq$ and evaluates Euler's phi function: $\phi(n) = (p-1)(q-1)$. This is the same as the number of numbers coprime to n. The sender then selects at random encryption exponent e such that $1 < e < \phi(n)$ such that e is coprime to $\phi(n)$ and computes the decryption exponent d such that $ed \equiv 1 \; mod \; \phi(n)$ using modular multiplicative inverse.

The public key is then (n, e) and the private key is (n, d). To encrypt the message m the sender computes $c = m^e \; mod \; n$. The decryption is similar $m = c^d \; mod \; n$. The beauty is in Fermat's little theorem, which states that $m^{\phi(n)} \; mod \; n \equiv 1 \; mod \; n$. Now, $ed \equiv 1 \; mod \; \phi(n)$, which means that $ed = k\phi(n) + 1$, and so $m^{(ed)} \; mod \; n \equiv m^{(k\phi(n)+1)} mod \; n \equiv 1^k m \; mod \; n \equiv m \; mod \; n$.

In practice, RSA requires random padding to protect against such attacks as chosen ciphertext attacks and making two identical plaintext produce various ciphertexts. Padding also ensures that the message size is multiple of encryption block-size. In practice, Optimal Asymmetric Encryption Padding (OAEP) scheme is used [].

It is good to know that if the message hashed and encrypted with private key, the result is a form of digital signature, since the sender cannot later deny that it was involved into encryption process. Digital Signature

Algorithm (DSA) is another example of asymmetric signature scheme and was specifically design for that purpose. Elliptic Curves another version of DSA algorithm.

Frankly speaking, symmetric signature schemes can be also used. For example, one can use one-time hash-based signatures to produce the secure digital signatures. Nevertheless, the application of these type of signature algorithms is rather impractical and finds little application in real-life settings.

2.1.3 Cryptographic hash functions

Mathematically, speaking hash function is special function that given a pre-image of an arbitrary size produces an image or hash of a fixed size, which is universally unique. Secure hash functions guarantee, to a certain degree, that the result of a hash function is irreversible. That it is, it should be extremely hard to find a pre-image, or original message, given a hash or fingerprint. Secure hash functions should be also collision resistant. In other words, it should be extremely hard, if not impossible at all, to find two different messages m and m' that will hash to the same value, i.e., $hash(m) = hash(m')$.

Secure hash functions are important in modern cryptography. For example, they serve as authentication tokens for transmitted message over the wire (useful, for example, in detecting message manipulation during transmission), they allow compressing the message before signing it with the digital signature algorithm, and they can be used to find the differences between the messages efficiently. The application area is of course broader than just these few examples.

Hash functions come in different flavors, but good ones should be computationally efficient and resistant to collisions. Today, hash functions such as MD2, MD4 and MD5 considered broken, as there are works that showed successful attacks. Briefly speaking, researchers successfully found collisions for this hash functions. Therefore, it is not recommended to use these hash functions in security applications. A more modern family of SHA hash functions also exists. For example, engineers recommend using SHA-256, SHA-512 and SHA-3 in modern applications, as no successful attacks were registered for these types of hash functions.

Hash functions pave a road for such a notion as authentication tokens when combined with a secret key in a special way. Examples are HMAC [], PMAC, CMAC which is based on AES cipher. For instance, by sending

an HMAC together with the original message one can make sure that the message will not be tampered during the transmission. In addition, if the message will be altered on the path to recipient, this fact will be flagged immediately during the verification process.

Hash functions are also useful in signatures. For example, one-time signature are using hash functions to construct an digital signature of a message. An interested reader can find more information about hash functions here [].

2.1.4 Key exchange protocols

Key exchange are important in modern systems as they allow negotiation of common key over insecure channel. Of course, RSA can be used to deliver a session key by encrypting it with the recepients public key, but specially crafted key negotiation algorithms exist in practice. Two bright examples are Diffie-Hellman (DH) and Eliptic Curve DH.

2.1.5 Post-quantum Lattice-based cryptography

Shor's algorithm, implemented on quantum computer makes certain computational problems (such as, factoring of large numbers and descrite logarithm problem) feasible in polynomial time. This shutters the security of the Internet, and so rigorous research was initiated to fill the gap. In what follows we discuss certain hard mathimatical problem on latticies and show the workings of the Learning With Errors (LWE) public key encryption scheme. In fact majority of NIST's candidates for post-quantum public key encryption algorithms are based on LWE.

A lattice is a mathematical structure which consists of integers in n-dimensions arranged in a structured lattice-like way. Mathematically, the lattice is defined as follows:

$$\Lambda(\mathbf{B}) = \{\mathbf{Bx}, \mathbf{x} \in \mathbb{Z}^n\}$$

where $\mathbf{B}$ is a matrix of basis vectors that generate the lattice. We should note that there exist large number of basis vectors, some are *good* some are *bad*.

A **closest vector problem (CVP)** on latticies, which is considered NP-hard, is not solvable even on quantum computers, can be defined as follows. Given a point $t \in \mathbb{R}^n$ and a lattice $\Lambda(\mathbf{B})$, the task is to find a closes point $\mathbf{Bx}$ on lattice:

$$\min_{\forall \mathbf{x} \in \mathbb{Z}^n} \|\mathbf{Bx} - \mathbf{t}\|$$

In practice the above problem is extremely hard to solve which makes lattice-based cryptography attractive to cryptographers.

From linear algebra we know that solving equation $\mathbf{Ax} = \mathbf{b}$ is simple using Gaussian elimination. However, if a random noise is added to the equation

$$\mathbf{Ax} + \mathbf{e} = \mathbf{b}$$

the problem is considered as hard as CVP on lattice. Solving the above problem directly relates to solving the CVP problem on lattice if the parameters are selcted carefully.

Given a matrix $\mathbf{A} \sim \mathbf{U}(\mathbb{Z}_\mathbf{q}^{\mathbf{nxm}})$, $\mathbf{s} \sim \mathbf{U}(\mathbb{Z}_\mathbf{q}^\mathbf{n})$ and $\mathbf{e} \sim \mathbf{D}_{\mathbb{Z}^\mathbf{m},\sigma}$ sampled from discrete (clipped) Gaussian distribution with parameter σ. We require that, the probability $P[e < q/4]$ is high (*i.e.* 99.99%) to ensure correct decryption of the message and to achieve required level of security. This can be achived by setting the parameter for discrete normal distribution carefully, *i.e.* $4\sigma < q/(4m)$.

To ensure that $\mathbf{re}$ is always less than $q/4$ the values are drawn with resampling or clipping. We require, further, that the parameter $\sigma = q/(16m)$.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ & & \dots\dots\dots\dots & & \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix} \tag{2.1}$$

$$\mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \\ \dots \\ s_n \end{bmatrix} \tag{2.2}$$

$$\mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ \dots \\ e_n \end{bmatrix} \tag{2.3}$$

Once the parameters are generated, we can compute $\mathbf{As} + \mathbf{e} = \mathbf{b}$. Then, the public key is $(\mathbf{A}, \mathbf{b})$ and the private key is $\mathbf{s}$. Derving $\mathbf{s}$ from $\mathbf{b}$ is a hard task at hand.

To encrypt the message $\mu \in \{0,1\}$ choose $\mathbf{r} \sim U(\{0,1\}^m)$. Then compute $\mathbf{u} = \mathbf{rA}$ and $v = \mathbf{rb} + \lfloor q/2 \rceil \mu$. The ciphertext is $(\mathbf{u}, v)$. To decrypt the message compute $v - \mathbf{us}$, if the result is less than $q/4$ output 0, if the result is larger than $q/4$ output 1.

The major disadvantage of lattice-based cryptography is the size of the keys and actual ciphertext. For example, security of LWE depends on two parameters n and q. By choosing $n = 512$ and $q = 2^{16}$, the size of ciphertext for a message of $k = 256$ bits long (for example, this is the size of the key for AES-256 symmetric algorithm), the size of the ciphertext will be $O(k \cdot n \cdot \log q) \approx 256 \cdot 16 \cdot 512$ bits or roughly wopping 256 KB. All in all the security does not come for free. Of course, there are way much practical implementations if LWE-based encryption algorithms, for example, the reader can take a look at Kyber [] which has practical implementation in TLS library.

2.2 Security protocols

2.2.1 Host Identity Protocol (HIP)

Internet was designed initially so that the Internet Protocol (IP) address has a dual role: it is the locator, so that the routers can find the recipient of a message, and it is an identifier so that the upper layer protocols (such as TCP and UDP) can make bindings (for example, transport layer sockets use IP addresses and ports to make connections). This becomes a problem when a networked device roams from one network to another, and so the IP address changes, leading to failures in upper-layer connections. The other problem is the establishment of an authenticated channel between the communicating parties. In practice, when making connections, the long-term identities of the parties are not verified. Of course, solutions such as SSL can readily solve the problem at hand. However, SSL is suitable only for TCP connections, and most of the time, practical use cases include only secure web surfing and the establishment of VPN tunnels. Host Identity Protocol, on the other hand, is more flexible: it allows peers to create authenticated secure channels on the network layer, so all upper-layer protocols can benefit from such channels. More on the protocol can be found in [**?**].

HIP relies on the 4-way handshake to establish an authenticated ses-

sion. During the handshake, the peers authenticate each other using long-term public keys and derive session keys using Diffie-Hellman or Elliptic Curve (EC) Diffie-Hellman algorithms. To combat the denial-of-service attacks, HIP also introduces computational puzzles.

HIP uses a truncated hash of the public key as an identifier in the form of an IPv6 address and exposes this identifier to the upper layer protocols so that applications can make regular connections (for example, applications can open regular TCP or UDP socket connections). At the same time, HIP uses regular IP addresses (both IPv4 and IPv6 are supported) for routing purposes. Thus, when the attachment of a host changes (and so does the IP address used for routing purposes), the identifier, which is exposed to the applications, stays the same. HIP uses a particular signaling routine to notify the corresponding peer about the locator change. More information about HIP can be found in RFC 7401.

2.2.2 Secure socket layer (SSL)

Secure socket layer (SSL) [**?**] and Transport Layer Security (TLS) are an application layer solutions to secure TCP connections. SSL was standardized in RFC 6101. TLS was standardized in RFC 5246. And was designed to prevent eavesdropping, man in-the-middle attacks, tampering and message forgery. In SSL the communicating hosts can authenticate each other with help of longer term identities - public key certificates. SSL is greate for building VPN tunnels and protecting upper layer protocols such as HTTP

2.2.3 Secure Shell Protocol (SSH)

Secure Shell protocol (SSH) is the application layer protocol which provides an encrypted channel for insecure networks. SSH was originally designed to provide secure remote command-line, login, and command execution. But in fact, any network service can be secured with SSH. Moreover, SSH provides means for creating VPN tunnels between the spatially separated networks: SSH is a great protocol for forwarding local traffic through remote servers.

2.3 L2, L3 and L4 tunneling

2.3.1 Virtual Private LAN Services (VPLS) L2VPN solutions

QinQ tunneling

MPLS tunelling

2.3.2 Virtual Private LAN (L3-VPN) security solutions

Multipoint to single point VPN

SSH tunneling

3. Results

In this chpater we are going to present the results that we have obtained throughout several years that we have spent building various systems. We start with the results for cryptographic library which we have implmented to boost the performance of AES and HMAC algorithms on Intel CPUs. We then present the results for complete HIP-VPLS architecture and present the lookings of the web interface which was used to configure the HIP switches. Finally, we present the design and implmentation of the hierarchical L3-VPN in Mininet emulator.

3.1 Hardware-enabled symmetric cryptography

Part of the work that we have done was related to porting parts of the code to pure C and special Intel CPU instructions. In this section we will describe our achivements in this direction.

For the benchmarkings we have selected three implementaitons. The first one was pure Python based. For that purpose we have used PyCryptodome library. The second implementation was a Python wrapper to C library that was using special Intel CPU instructions to boost the AES and SHA-based HMAC operations. The third implementation was pure C library which was using Intel NI instructions. The results for AES-256 and HMAC operations for varying block sizes is shown in Figure **??**. The plots show the average running time in miroseconds with the 95% confidence intervals.

What related to cryptographical operations. For standard packet of size 1500 bytes we have compared the performance (combined HMAC and AES-256) and it turned out, on one hand, that implementation of cryptography in pure C with CPU instructions is 12.1 faster than pure Python implmentation. On the other hand, Python implemenation with bindings

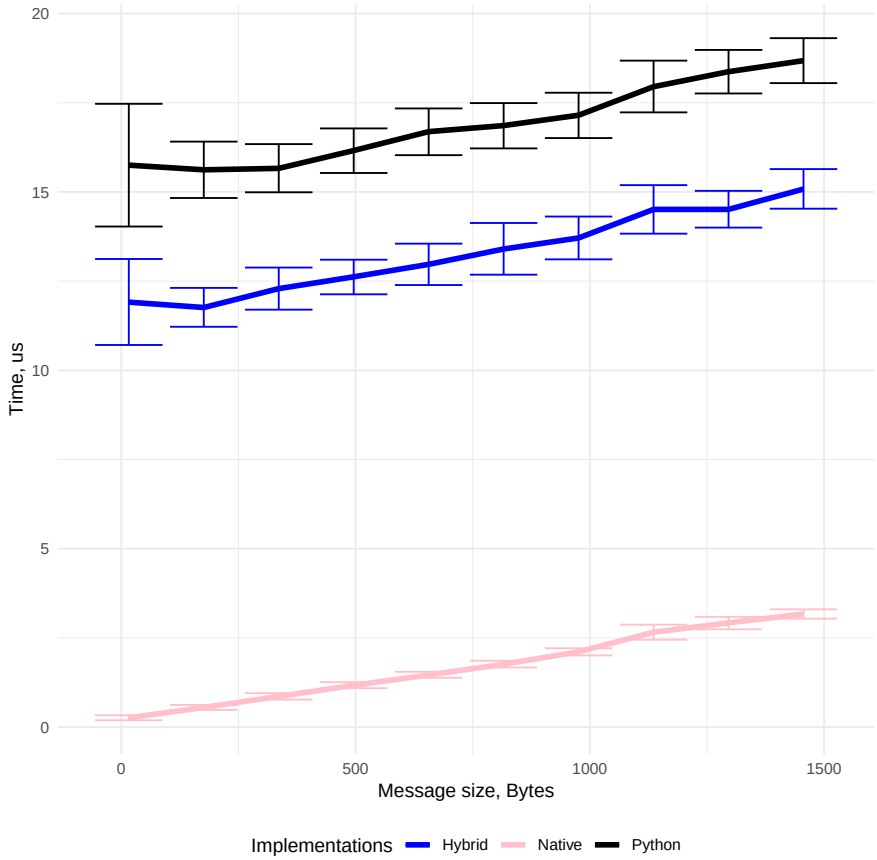

Figure 3.1. AES-256 encryption (microseconds)

to C library demonstrated performance which was 2.3 faster. By making back of the envelop calculations we predict that Python implmentation can achive roughly 60 Mbits/s in upload and download directions cumulatively. In fact this exactly the result we get in our experiments.

Statistics	Upload (Mbits/s)	Download (Mbits/s)	Lattency (ms)
Sample mean	46.1	48.2	5.0
Sample std	7.1	2.3	0.19
Sample median	44.8	48.8	4.9
Sample min	14.3	40.0	4.6
Sample max	61.3	50.4	5.4

Table 3.1. Performance of HIP-VPLS on Intel N95 CPU

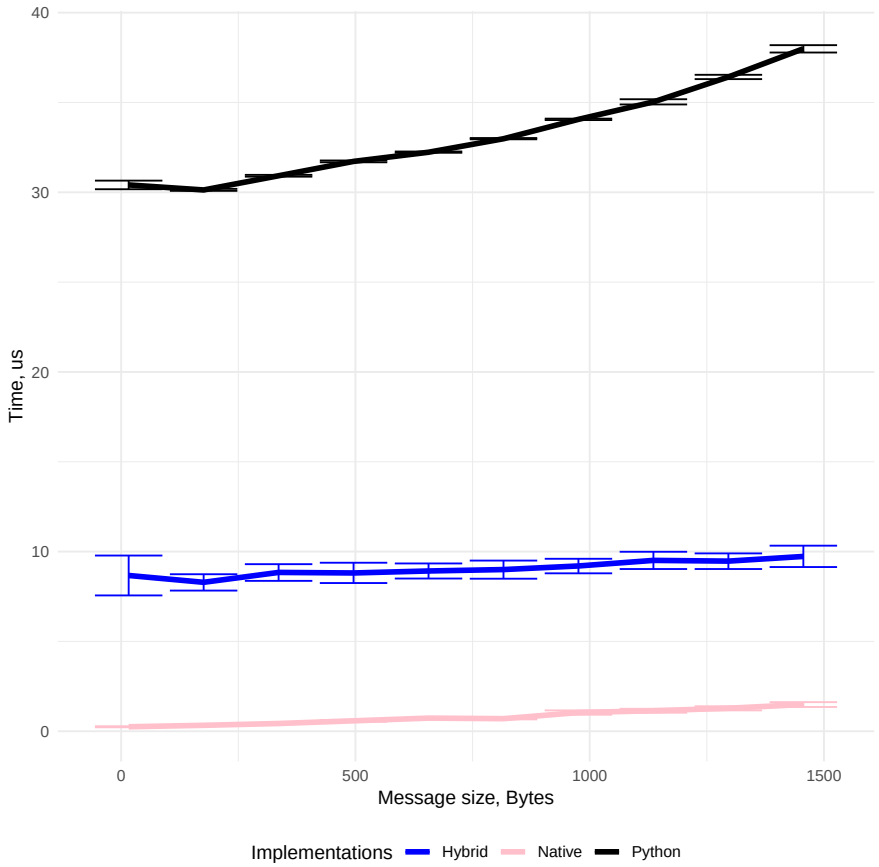

Figure 3.2. HMAC calculation (microseconds)

3.2 Host Identity Protocol based VPLS

3.3 Scalable multipoint to multipoint VPN using HIP protocol

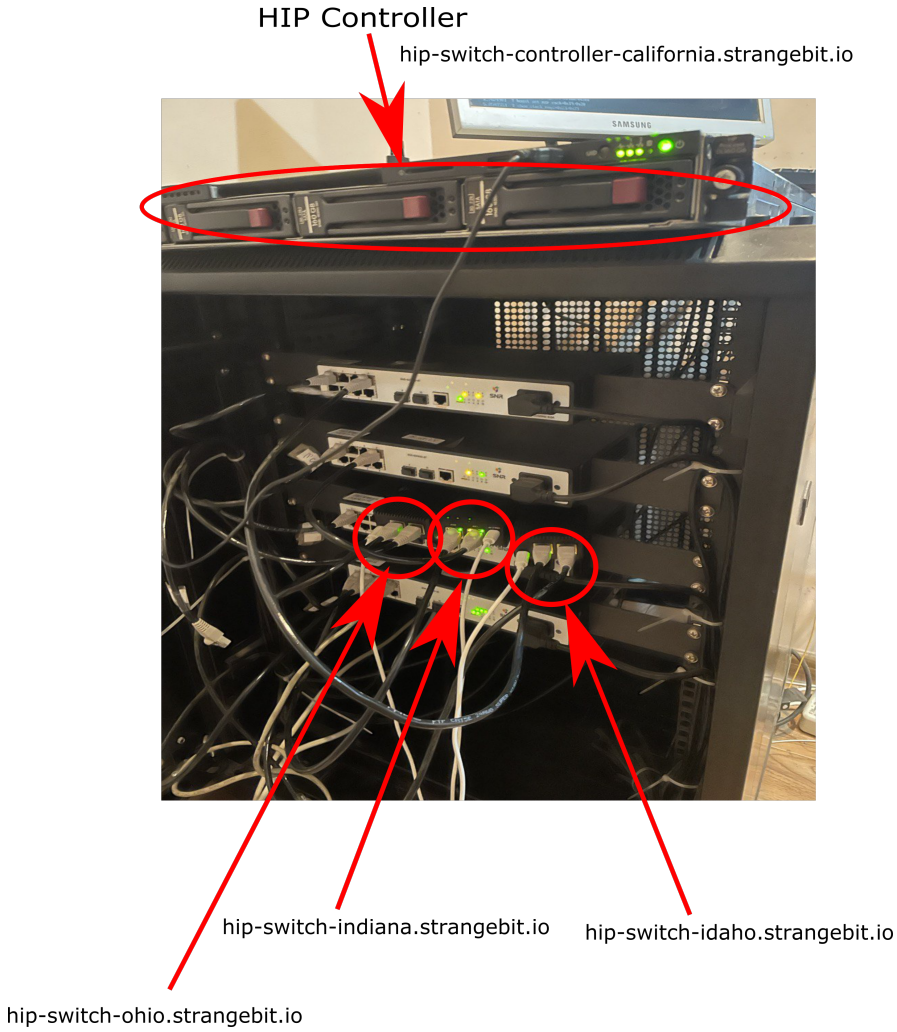

Figure 3.3. Testbed

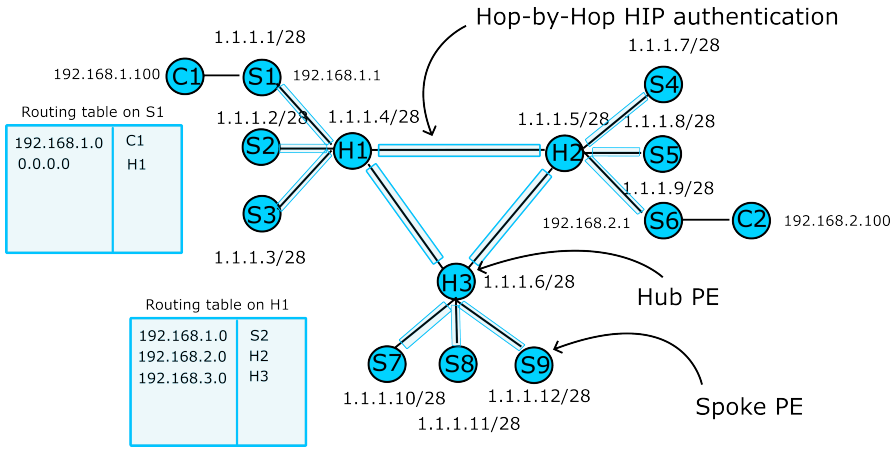

Figure 3.4. HIP-based L3-VPN in Mininet

4. Conclusions

Bibliography