TDDE21 - HIPv2

Hedlin, Johan
johhe911@student.liu.se

Jungmalm, David
davju247@student.liu.se

Kahlström, Joakim
joaka568@student.liu.se

Wrede, Simon
simwr872@student.liu.se

December 18, 2020

1 Introduction

1.1 Project purpose

The purpose of this project has been to extend the Host Identity Protocol (HIP) implementation OpenHIP[1] with support for customizable hash functions[2], new cryptographic algorithms[3], and hierarchical Host Identity Tags[4] (HITs). There have also been new automated tests created for the HIP Base Exchange (BEX) and IP address mobility since the existing tests were based on older software that would not easily run on modern systems.

2 Background

2.1 Host Identity Protocol (HIP)

The Host Identity Protocol offers a more persistent method of addressing than the Internet Protocol (IP), where the address of a host changes depending on which network it is connected to. With HIP, addresses are calculated using public-key cryptography and uniquely identify a single host independently of its current network location. The public key-based identities also provide a stronger guarantee that the contacted host is the one responding, as others would not be able to create valid packets with a fake source address. HIP uses a four-way handshake called the Base EXchange (BEX) when establishing a connection to another host. The second packet in this exchange, which is transmitted from the contacted host to the initiating one, also contains a challenge which requires some computation to solve. The contacted host will not maintain any connection-related state variables until this challenge has been solved by the initiator, which avoids denial-of-service attacks on the computationally expensive key exchange.

2.2 Project goals

For this year's project, three Internet-Drafts were to be implemented:

- **draft-moskowitz-orchid-cshake**
  This draft provides an alternate method of generating the Overlay Routable Cryptographic Hash Identifiers (ORCHIDs) (used in the creation of HITs) which uses the variable output length hash function cSHAKE to produce a given number of output bits instead of truncating the hash function output. In addition to this, there is also a new parameter called "Additional Information" which, as the name suggests, can be used to include up to 32 bits of arbitrary information in the HIT without

[1] `https://bitbucket.org/openhip/openhip`
[2] `https://datatracker.ietf.org/doc/draft-moskowitz-orchid-cshake`
[3] `https://datatracker.ietf.org/doc/draft-moskowitz-hip-new-crypto`
[4] `https://datatracker.ietf.org/doc/draft-moskowitz-hip-hierarchical-hit`

inducing a significant risk of hash collisions. The only current use of this is the new EdDSA/cSHAKE HIP suite from the new-crypto draft described below, as well as the hierarchical HITs that were also implemented during this project.

- **draft-moskowitz-hip-new-crypto**
  The new-crypto draft mainly introduces the use of the Edwards Elliptic Curve Digital Signature Algorithm (EdDSA) for use as Host Identities (HIs) and for authenticating the HIP BEX. It also specifies new elliptic curves (Curve25519 and Curve448) for use in the Diffie-Hellman key exchange, a lightweight algorithm known as Keyak that can be used in place of AES to encrypt parts of the BEX, and the use of the Keccak Message Authentication Code (KMAC) function as a potentially more efficient key derivation function.

- **draft-moskowitz-hip-hierarchical-hit**
  This draft places additional registration information inside the HIT through the method specified in draft-moskowitz-orchid-cshake. This information consists of two 16-bit fields that identify a Hierarchical HIT Domain Authority (HDA), which provides services for identification and lookup to HIP clients, and a Registered Assigning Authority (RAA), which manages a set of HDAs. Together with the HI hash, these fields form a unique identity which can be looked up and verified by contacting the specified HDA. This solves the scalability problems that would arise with a flat address space, as storing up to billions of entries in a single database could be difficult.

As these are working drafts of new standards, it is stated in the beginning of each document that they could be changed or abandoned. This has been the case in the New Crypto draft, where parts of the draft was abandoned and others updated. However, the changes did not affect this project that much and the draft could still be implemented.

In addition to these drafts, the project group also aimed to improve the existing automated tests. These were written for Python 2, which has already reached end-of-life, and also target an old version of the Common Open Research Emulator[5] (CORE) network simulator which is difficult to install on modern systems due to library conflicts. To make it easier to automatically test OpenHIP on modern systems and to maintain it in the future, the tests will be rewritten using Python 3 and the latest version of CORE.

2.3 Relevant files

src/util/hitgen.c Used for generating private keys and Host Identity Tags (HITs), storing these in XML format to be used when starting OpenHIP.
src/util/hip_util.c Miscellaneous helper functions and logging utilities. Also contains methods for generating and validating HITs.
src/include/hip/hip_types.h Contains data structures like _hi_node which contains information like the HIT and information used to generate the HIT.
src/protocol/hip_input.c Logic for incoming packets.
src/protocol/hip_output.c Logic for outgoing packets.
src/test/hipnode.py An extension of CORE's CoreNode class to represent a computer running OpenHIP.
src/test/testcasecore.py Skeleton class to use with Python's `unittest` library.

3 Method

For this year the students had to implement three new features into the OpenHIP project as mentioned in Section 2.2. The first thing that was done was to divide the work between the members in the group.

- New Crypto: Johan & Joakim
- New ORCHID: David & Simon

[5]`https://www.nrl.navy.mil/itd/ncs/products/core`

- New Tests: Simon
- Hierarchical HIT: Everyone when the previous parts are finished

In the beginning of the project there were five members, one of which was assigned to New Orchid, but due to that member dropping the course Simon was later assigned to this part of the project. As it was determined that hierarchical HITs were based of the new ORCHID draft, this part was not initially assigned to any member.

3.1 Developing Environment

The operating system used during the development has been Linux. Two students have been using Ubuntu 20, one has been using Ubuntu 18 and one has been using Arch Linux and Debian unstable. This has worked quite well and the group has been able to help each other when there have been problems. Not all of us had Linux already installed but was solved with a virtual machine or by using Windows Subsystem for Linux (WSL).

Visual Studio Code was mostly used during development, which also gave the ability to collaborate together remotely in an easy way with its Live Share extension functionality.

To be able to test the protocol, the CORE network emulator was used. It could be set up with both GUI and Python scripts. The GUI was good to visualize the communication and the network structure but took a few steps to set up properly each time. The Python scrips were used to automate the testing procedure and test changes in the code a lot quicker than before.

4 Results

4.1 Before half-time presentation

One of the main challenges before the half-time presentation was to get acquainted with the code and read the drafts of the features that were supposed to be implemented. As none of the students had worked with HIP before, it also included reading the HIPv2 RFC[6].

It was also a challenge to completely work remotely, which was required due to Covid-19. To counter the loss of not being able to work face to face, more regular meetings were scheduled in combination with time where project work could be done together.

At the point of the half-time presentation the new-crypto draft was partially implemented, such that the EdDSA key generation and signing/verifying, the KKDF key derivation function, new Diffie-Hellman curves, and the Keyak cipher were working. Issues with the HIT suite not being correctly set for both parties of the connection prevented the entire base exchange from completing, but parts of it were working.

Work on the orchid-cshake draft consisted mainly of reading through the OpenHIP code, the draft itself, and the HIPv2 RFC. Some preliminary bits of code and helper functions were produced, but HIT generation was not yet possible. Since the hierarchical HITs depended on the new ORCHID generation process, no work could be done in this area apart from studying the relevant internet-draft.

4.2 Plans for the second half

The project group decided to use the service Monday.com[7] to help with organizing the project and get an overview of the project and remaining tasks. This also helps due to everyone working remotely from home. A more concrete time plan was also established and entered here so that everyone could at a glance see if the project was on track time-wise, and what areas needed extra development efforts.

The main focus was to wrap up development on the new-crypto draft within 2 weeks, and in parallel get a working implementation of the orchid-cshake draft completed within 3 weeks. This would leave 3 weeks to work on hierarchical HITs plus the final report and presentation.

[6] `https://tools.ietf.org/html/rfc7401`
[7] `https://monday.com`

4.3 New crypto

The implementation of the new-crypto draft went mostly smooth, with steady progress being made each week. But as with any software development project, there were a couple of issues that halted progress at several points. One of these was the fact that the existing code did not work when attempting to compile it and establish a connection between two hosts. Since no work could be done without having a known-good state to start from, this had to be resolved first.

After a few days of reading through the codebase and debugging, it was discovered that the IPv4 checksum calculation in OpenHIP had been slightly modified in an attempt to fix a compatibility issue. In combination with IP checksum offloading being disabled on CORE's virtual test network, this caused all transmitted packets to be dropped at the receiver due to checksum mismatch errors.

During the implementation of the EdDSA HI algorithm, there were also some issues with the negotiated HIT suite not being available during parts of the base exchange. Since parts of the base exchange involve signing and verifying the transmitted data, the algorithm used for this purpose must be known before the received data can be verified. However, the variable containing the chosen suite was sometimes set too late or not at all. When not set, the variable had a value of 0, which is a reserved value in HIP. This value should never be used, and as such the base exchange should have been aborted with a notice that an invalid HIT suite has been selected. Instead of this, OpenHIP contained several fallbacks that selected the RSA/DSA suite when no valid suite was supplied, which had the effect that the RSA/DSA suite would be chosen no matter what suite was negotiated during the base exchange. To correct this, the order in which the R1 packet parameters are parsed was changed in order to make the selected suite available in time for the signature verification step. There were also warnings added which are displayed whenever a fallback is triggered.

Some other problems that were encountered include:

- When parsing the R2 packet, the HIP_MAC_2 parameter was moved out of the received packet and then copied back, but this process failed to account for the padding inside the parameter. This caused EdDSA signature verification to fail, but RSA/DSA seemingly worked nonetheless.
- The HIT verification process also relied on the HIT suite ID variable being set, which created problems when this variable had not yet been assigned a value. Since the HIT contains the OGA ID, which in this case is the same as the suite ID, this value is now extracted from the HIT being verified instead of relying on the suite ID variable.
- The hitgen utility, which generates the key pairs used for the HI and the HIT, did not put the used suite in the OGA ID field of the generated HIT. Due to the previously mentioned fallbacks, this still worked for RSA but was not compliant with the ORCHID generation process.
- The MAC/HMAC calculation process in *build_tlv_[h]mac()* contains the code segment `hmac_md[hmac_md_len - sizeof(hmac->hmac)]`. This indexes hmac_md out of bounds if hmac_md_len is smaller than 64, which it is for hash algorithms with less than 512 bits of output. Indexing this array out of bounds seems to possibly leak some stack data to the other network node, which aside from possible security concerns could also lead to interoperability issues with other HIP implementations.

Some of the problems encountered were design related, where a decision on the correct course of action could not be made without input from others. Questions were sent to Gurtov who forwarded them to the right people. Within a few days there were a few replies with possible solutions, allowing the project to proceed.

4.4 New ORCHIDs

The purpose of the new ORCHID draft was mainly to allow for hierarchical HITs. As mentioned in Section 2.2, the new ORCHID splits the previously 96 bit large hash into a hash and an field containing an arbitrary amount of information up to 32 bits. For HHITs the information field is 32 bits which is exactly 4 bytes. Since the HIT is stored in a 16 byte array (128 bits) the insertion is quite straight forward, but if the arbitrary information is of a size that is not divisible by 8, one would need to use bit shift operations in order to correctly align the information field and the hash field. As can be seen in Figure 1, when using HHITs one can simply place additional information at index 4 and the hash at index 8. However, as mentioned it is

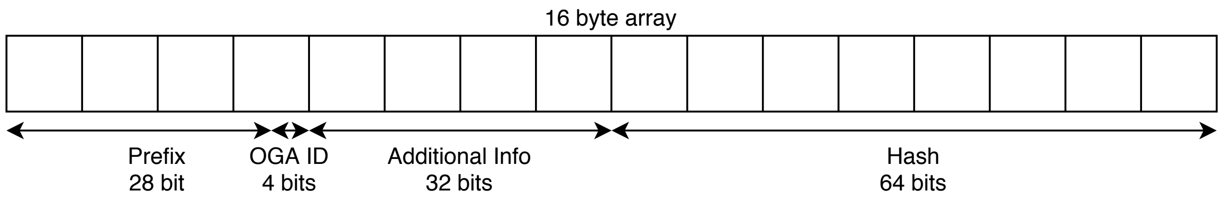

Figure 1: Hierarchical HIT layout

not that straight forward when additional information is for example 30 bits, then it is necessary to bit shift the hash in order to have it in the correct place. For this reason, it was decided that instead of the user defining how large additional information should be, the user would specify a separate flag telling hitgen to use HHITs in the generation by using a flag called -hhit <info> where info in this case needs to be exactly 4 bytes in order to behave as expected, else truncation will occur. Since the new ORCHID draft includes new input parameters to the hash functions, a decision to make a clear separation of generation method for flat HITs and HHITs was made.

4.5 Hierarchical HITs

There were some different ideas on how one would configure OpenHIP to use HHITs insted of flat HITs which is the default setting. It was determined that as long as there is no way of getting an assigned hierarchical ID (HID) but only configure one manually this would be done in the file called `hitgen.c` as mentioned in Section 4.4. Since the hierarchical-hit draft mentions that HHITs should use a separate prefix, one could assume that for any use case of the additional information, there should be a separate prefix. This allowed for a split in the HIT generation based on the prefix. When generating a HIT, using the -hhit flag, one also stores information about the information field use case. Based on this, a separate prefix is used, which can later be used for validating HITs since the prefix will tell what HIT generation method has been used.

4.6 Tests

The already existing tests for OpenHIP were written in Python 2 and used a very old version of CORE. It was therefore difficult to run them or write new tests since Python 2 was deprecated in 2020. As an additional goal for 2020, the tests were rewritten in Python 3 and utilizing a modern CORE version.

The tests were also written to make use of Pyhton's built-in libraries. Previously the tests required external dependencies and performed a lot of string operations on XML-files. The new tests use the `unittest` library as a test runner and the `xml` library to handle all OpenHIP configuration files. The results are an updated test suite with modern programming standards. Additional tests can easily be added if one is familiar with CORE's Python libraries.

4.7 Final result

All the yet possible features from all 3 proposed drafts were able to be implemented during this project. There are still some things that could not be implemented yet, and those are mentioned in Section 6.

5 Discussion

Despite everyone working remotely during the entire project the work on all the tasks proceeded efficiently. Dividing the work into smaller groups also reduced the amount of code that each group had to read and familiarize themselves with. During the weekly online meetings the common parts of the code were discussed, and pointers were given to the other group on which parts were the most interesting out of "our" parts of the codebase.

Using a mix of different Linux distributions creates the risk of running into issues that only affects one project member, but could also decrease the risk of accidentally creating code that happens to work in the specific environment used but fails in another.

6 Future work

6.1 Hierarchical HITs

There are still a few things in the hierarchical HIT draft by Moskowitz et al. that needs to be implemented before this part of the project is usable in real world scenario. They are described below.

6.1.1 Missing HHIT prefix

The first thing that is missing is an official assignment of the HHIT specific prefix that is mentioned in the draft. Since there were no allocated prefix[8] at the date of implementation, a temporary prefix of 2001:30::/28 is used. This prefix is stored in `hip_types.h` and it is also mentioned in the code that this is a temporary value that should be replaced as IANA allocates a specific address for this purpose.

6.1.2 The Hierarchy ID (HID)

As mentioned in Section 2.2, the HHIT make use a field of arbitrary information that have a maximum size of 32 bits. For the purpose of HHITs this field is always 32 bits and the draft mentions a further division of these 32 bits into the following:

- 16 bit Registered Assigning Authority (RAA)
- 16 bit Hierarchical HIT Domain Authority (HDA)

Exactly how a HID should be retrieved is deemed to be out of the scope of the draft. Therefore the code can not automatically get a HID, but only assign one manually. This is something that needs to be implemented later for this functionality to work as intended.

6.2 New crypto

The key length for Keyak is not specified in the new-crypto draft, and was temporarily set to 256 bits. This should be changed in `src/include/hip/hip_proto.h` when a proper key size is chosen by the draft authors. The KMAC result in *build_tlv_mac()* is currently truncated to 160 bits (the same as for HMAC), but at least one implementation[9] states that the KMAC output should not be truncated when used like this. A better option would be to directly specify that 160 bits of output is desired, and let KMAC output this size directly. This would also avoid the issue mentioned in section 4.3 where the *hmac_md* array is indexed out of bounds.

7 Advice

This section gives advice on how future students can easily get started developing OpenHIP. Alternatives for development tools to different systems are discussed and several tips are brought up.

7.1 Operating system

OpenHIP can be installed on Linux, Windows and macOS. This section will go over the existing options for developing on either system.

[8] `https://www.iana.org/assignments/iana-ipv6-special-registry/iana-ipv6-special-registry.xhtml`
[9] `https://www.cryptosys.net/manapi/api_kmac.html`

7.1.1 Linux

Linux operating systems are supported natively by both OpenHIP and CORE. No further configuration should have to be made except for installing any missing packages with your system's package manager.

7.1.2 Windows

OpenHIP can be installed and used on Windows, CORE on the other hand requires a Linux kernel. There are several options for running a Linux distribution in a virtual machine on Windows such as Oracle's VirtualBox[10] or VMware[11]. After installing a virtual machine the installation process is the same as on a regular Linux system.

Windows does however have a better alternative to third party VM software, Windows subsystem for Linux 2 (WSL2). Microsoft has stellar instructions on how to install it[12]. WSL2 has a full Linux kernel (as opposed to WSL1 which does not) and can therefore use CORE and OpenHIP without problems.

To run graphical applications like CORE's GUI you need an X Server for Windows and also instruct WSL2 how to use this. VcXrv[13] is recommended but another alternative is Xming[14]. In WSL2, open `~/.bashrc` and add the code seen in Listing 1 to correctly set the `DISPLAY` environment variable. Remember to reload the file with `source ~/.bashrc`.

```
export DISPLAY=$(cat /etc/resolv.conf | \
                 grep nameserver | \
                 awk '{print $2}'):0.0
export LIBGL_ALWAYS_INDIRECT=1
sudo /etc/init.d/dbus start &> /dev/null
```

Listing 1: Setting the `DISPLAY` environment variable in `~/.bashrc`.

When starting the X Server make sure to tick the checkbox `Disable access control`. You may also have to disable the firewall for WSL only. This can easily be done in an elevated PowerShell prompt using the command seen in Listing 2.

```
Set-NetFirewallProfile \
    -Name public \
    -DisabledInterfaceAliases "vEthernet (WSL)"
```

Listing 2: Disable the firewall for WSL in PowerShell.

7.1.3 macOS

OpenHIP will run on macOS but CORE will not. An option is to install a virtual machine using a Linux distribution and follow instructions for regular Linux. Oracle's VirtualBox[15], VMware[16], and Parallels[17] are all VM alternatives for macOS.

[10] https://www.virtualbox.org/
[11] https://www.vmware.com/
[12] https://aka.ms/wsl2-install
[13] https://sourceforge.net/projects/vcxsrv/
[14] https://sourceforge.net/projects/xming/
[15] https://www.virtualbox.org/
[16] https://www.vmware.com/
[17] https://www.parallels.com/

7.2 CORE

Gurtov has published instructions on how to install CORE on the course webpage. But perhaps an even easier and up-to-date way is to use CORE's automated install script[18]. It is recommended to install locally, i.e. `bash install.sh -l`. This will properly setup CORE's Python libraries and you can execute Python scripts as normal with `python3 script.py`. Otherwise you would have to start Python scripts using CORE's virtual environment with `core-python script.py`. Proper installation of CORE's Python libraries also allows for intelligent code completion in your IDE.

7.3 OpenHIP

This section guides you how to best navigate the OpenHIP source code.

7.3.1 Repository

OpenHIP is installed easily by following the instructions in the README[19]. Gurtov has also published instructions on how to test your OpenHIP installation on the course webpage. As mentioned by previous students, these instructions contain a typo where `mkdir $SESSION_DIR/n2.conf/usr.local.usr.local.etc.hip` should be run as `mkdir $SESSION_DIR/n2.conf/usr.local.etc.hip` instead. With CORE installed and using OpenHIP branch `hipv2` one can also run the new tests (`python3 test/test_switch.py`) to confirm that OpenHIP is working.

7.3.2 IDE

It is recommended to use an IDE with C language support to allow for type-hinting, intelligent code completion and code navigation. The IDE used in this project was Visual Studio Code.

7.3.3 Doxygen

One very useful tool for discovering previously unknown source code is Doxygen. It can provide information in the different data structures, relations between functions and provide dependency graphs between files. In the early stages of the project, this can be really useful instead of manually trying to figure out which functions calls which and what relation the different files have. See Figure 2 for an example of how Doxygen can visualize how the function named *hip_parse_I1* is called and Figure 3 for an example of which other functions it calls.

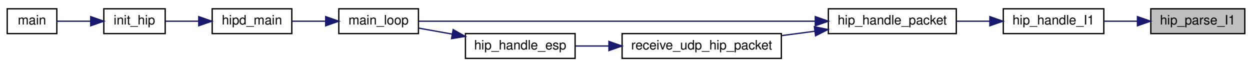

Figure 2: Call graph for hip_parse_I1

[18]https://coreemu.github.io/core/install.html#automated-install
[19]https://bitbucket.org/openhip/openhip

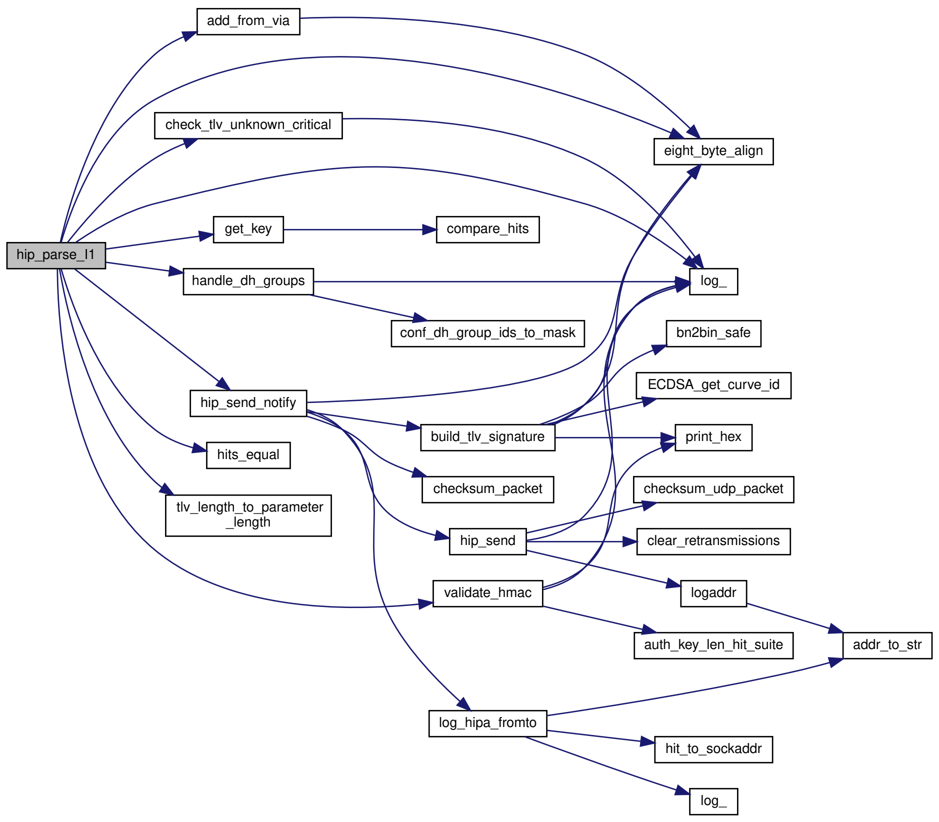

Figure 3: Functions called from hip_parse_I1

7.3.4 Generating EdDSA host identities (HIs) and hierarchical HITs (HHITs)

Using the `hipv2` branch, EdDSA HIs can be generated by following the instructions referenced in Section 7.3.1, but substituting the `hitgen` command for `hitgen -type EdDSA -suite 5`. To enable the EdDSA suite, the line `<suite>5</suite>` needs to be added before the existing entry for suite 1 in the `<available_hit_suites>` section of `/usr/local/etc/hip/hip.conf`. The Keyak cipher and the new Diffie-Hellman groups are enabled, but set to a lower priority than the previously existing options. This means that they will never be used when initiating a connection, but incoming connections using these algorithms are supported. The file `src/protocol/hip_main.c` specifies which options that should be enabled and used by default, and can be modified if this behaviour is not desired.

To generate HHITs, simply pass the argument `-hhit INFO` when running the `hitgen` command, where the string "INFO" can be replaced with four arbitrary characters corresponding to the desired 32 bit hierarchy ID.