Distributed Cryptography with IoT devices

Isaac Barrett
Github: hornetfighter515
hornetfighter515@protonmail.com

Lachezar Todorov
Github: Todorov Lachezar
lxt9427@rit.edu

December 2021

1 Introduction

Distributed cryptography is a term that refers to cryptographic operations carried out by a number of mutually distrusting parties in order to reach a common goal. An attacker who obtains information from a certain number of parties cannot recover all of the necessary pieces or breach the security of the system [2][4][9]. Prior works in this field have focused on protecting the secret keys, on the premise that with symmetric-key encryption the security of a large amount of data can be simplified to the security of one small key [1]. Symmetric key cryptography is often chosen since it stresses devices less than asymmetric cryptography, and is thus faster and more computationally efficient. However, many of these assertions are based on old technologies, where fewer hardware and software optimizations are made for asymmetric cryptography[11].

Public key cryptography is based on the idea of the one-way trapdoor function. These functions are infeasible to undo unless a specific factor, or "key," is known. In most modern public key cryptography, there is a pair of keys: the public key, which is used to encrypt a message, and the private key, which is used to decrypt. Under normal conditions, this communication is between two parties. We propose a solution which has third parties who do not need to be trusted due to the nature of the system, but still increase security for the end user while at home. Research is being conducted to allow for easier modification of encrypted data without ever having to decrypt it [7]. This could mean huge advances for public key cryptography. However, as it stands, there is no way to easily do this.

Internet of Things (IoT) is about communicating data between objects that have computing capabilities over the internet [5][12]. These devices are progressively being used to store secrets that are used to authenticate users or provide security. Many IoT devices are not equipped with necessary hardware to store secret information, and when they are equipped, they provide little to no configurability for their applications. Thus we plan to make use of the fact that many people own smart devices (smart phone, smart watch, smart TV, etc.) to distribute the work among them (rather than keeping it on a single device) to enable multi-device cryptographic functionalities [1]. Given the limitations and low power of IoT devices, we seek to find a solution which may be practical to implement.

A framework will be built in which IoT devices can participate in user transactions in order to increase their personal security over the internet. Similar to how a VPN provides a twice-encrypted shell for a user (once for the VPN itself, and again for any secure connections a user makes), the IoT network will provide a user additional security so long as they are in their own home. In the age of work-from-home culture, this is more beneficial than ever.

Our main goal of this implementation was to use the IoT devices to add an extra layer of security to the standard public-key cryptographic method. We structured our implementation in a way to remedy the greatest weakness with any cryptographic method: the compromising of a single key, whether private or asymmetric. The concept of protecting private key has already been researched and implemented [1][2], so we took a different approach to protect the data via duplicity. In standard public key cryptography, the private key acts as a single point of failure. If an attacker copies the private key without the intended recipient's knowledge, the attacker does not need persistent access to the victim's machine to continue to decrypt messages; they only need to intercept messages intended for the recipient. Our approach seeks to solve the issue of private or symmetric key compromise, such that even with owning the key of the intended recipient, a message could not be intercepted and decrypted over the general internet.

The theory behind our work is difficult to test, since it would rely on penetration testing with malware advanced enough to get access to memory. Instead of focusing on the theory itself, this paper will focus on the implications of implementing such a system, such as extra network noise, increased sending times, and increased computational demand. Our scope for the implementation is encryption and decryption. Although authentication is important, this project does not directly address the complexity of implementing a full and in-depth protocol.

2 Implementation

Our implementation has a few important assumptions for it to work optimally, but they are assumptions that can be made in good faith. The first is that there are other solid security measures in place that will allow an intrusion to be non-persistent. A persistent attack would allow the attacker to circumvent encryption in other ways, such as simply accessing the plaintext after decryption. It would also allow an attacker to still have a presence on the network, meaning any traffic on the local network could be decrypted if the keys were compromised during the agreement/sharing process. The second is that not all IoT devices would be compromised. It is well known that IoT devices are often incredibly vulnerable due to their decreased computational power [12], and so this experiment assumes the standard integrated smarthome use-case scenario: a higher-protected subnetwork for the IoT devices. In practice, this is usually accomplished by daisy-chaining routers and placing the smart network behind a firewall. The third is that the intended recipient is on their home network. Without being at home on the same local uncompromised network, this approach's overhead would be fruitless.

Based on these assumptions, the intended recipient's security can be significantly increased if the message sent to her has some extra work done on it. Below is the process taken in the form of a multi-step list.

1. The sender writes a plaintext message.
2. The sender encrypts that message with the intended recipient's public key.
3. The sender chunks the ciphertext into n pieces.
4. The sender encrypts each of n chunks with the public key of one of each of n IoT devices, respectively.
5. The sender delivers each message to the recipients network, specifically to the smart hub.
6. The smarthub gets each message and sends it to the respective IoT device.

7. The IoT device decrypts the chunk.
8. The IoT device delivers the now-decrypted chunk back to the smarthub.
9. The smarthub sends the decrypted chunk to the recipient.
10. The intended recipient waits until all chunks are received back from the smarthub.
11. The recipient arranges the chunks into the proper order, then decrypts using their private key.
12. The recipient views the plaintext message.

The architecture of our system is best described through example. For this example, Alice is at home. She has a collection of her personal devices, which she actively and directly interfaces with. Those devices are on her main network. Then, her IoT devices are placed on her protected subnetwork. Her protected subnetwork has a firewall between its router and her general network's router. Of course, since this is a home set-up, she does not have state-of-the-art security protecting her subnetwork. However, she does have a relatively standard configuration from a professional smarthome integrator company. Alice is also a responsible user, and if she were hacked, she would have the malware removed from her computer. However, the programs she uses may not necessarily regenerate her private keys used in communications. Under normal conditions, this leaves her in an incredibly vulnerable state. If an attacker had access during her previous key-generation and managed to compromise her private key and all other keys, the attacker could intercept and decrypt all private communications intended for Alice, compromising confidentiality. However, our solution seeks to secure Alice's communications even with complete compromise of her private keys stored on her personal machine.

Effectively speaking, each IoT device's private key becomes a part of Alice's private key. Alice's true private key will be called a, and there are n number of IoT devices. Even if the attacker gains access to a, and even with illegal access to $a+(n\text{-}1)$, so long as any singular IoT private key is not compromised, Alice's communications will remain as secure as the algorithm used to encrypt the messages.

Due to the nature of the system, the only places that could ever rebuild the true ciphertext are Alice, the smarthub, and the network's router. But as discussed before, if Alice's machine is compromised, an attacker needs not waste his time decrypting. One major flaw in this system is that if the smarthub or the network router are compromised, the attacker would still be able to reassemble the ciphertext, and if Alice's private key was truly compromised, the attacker would still be able to decrypt said ciphertext. This is why we must assume non-persistent access to the network.

3 Testing

Two devices were compared to decrypt equal-length messages synchronously. The first, called the "Simulated Device on High Power PC," or PC for short, was on a high end gaming PC. The hardware included is a NVIDIA GeForce RTX 2080 Ti, an AMD Ryzen 3950x (with default settings), and most importantly for our use case due to decryption logic chips, an ASUS Tuf Gaming X570. This combination of high-end hardware allows for lightning fast asymmetric decryption times, as well as a fast network transmit time due to being hardwired. The other device involved was a Raspberry Pi 3B+. This is a more realistic embedded-type device, and is interchangeably called

Device	PC	Pi
Operating System	Arch Linux	Raspbian 11
Kernel	5.15.5-arch1-1	5.10.630v7+
Central Processing Unit	AMD Ryzen 9 3950X	BCM-2835
CPU Clock Speed	3.5 GHz	1.2GHz
Memory	32068 MiB	922 MiB
Graphics	NVIDIA GeForce RTX 2080 Ti	Integrated
Extra Logic Chips	Present	Absent
Connection Method	Ethernet	WiFi

Table 1: Presents information about hardware and firmware of each system being used.

the IoT device from here forward. Although it is not a perfect analogy to the embedded devices we see being produced today, since it has a full-fledged operating system, it is close enough for the purposes of our research.

The PC, for this experiment, is meant to be a hyperbolic representation of the average user's setup, purposely chosen to compare to an embedded device for its high speeds. It is meant to be as unfair to the IoT device as possible to show the limitations of embedded technologies in this context.

The Python3.9 RSA library was used to encrypt messages [10], which were sent over websockets [6] [8]. SSL was foregone in this experiment for the sake of scope and brevity. The goal is to test the efficacy of our implementation, not the efficacy of SSL. Additionally, we are using a separate encryption method. The RSA library is known to be slow, since it is implemented in pure Python, which will always have the limitation of the global interpreter lock. However, for our purposes, this works to allow the differences of time to be shown measurably. A keylength of 2048 was used.

64 messages were sent from a smartphone over the internet to the intended recipient. For our purposes, this means that 128 different messages were sent to the target's network, since our experiment used n=2. Rather than try to measure times spent on extra devices, and rather than waste time measuring operations with constant-time complexity from the smarthub, times were measured by how much they augmented, or added to, the total message sending time. A timer was initiated on step 6 of the process above, and stopped on step 8. This data was measured from the point of view of the smarthub, then tracked in a file. Augment times were not measured for encryption from the point of view of the sender, since this would be highly dependent on the hardware the sender is using.

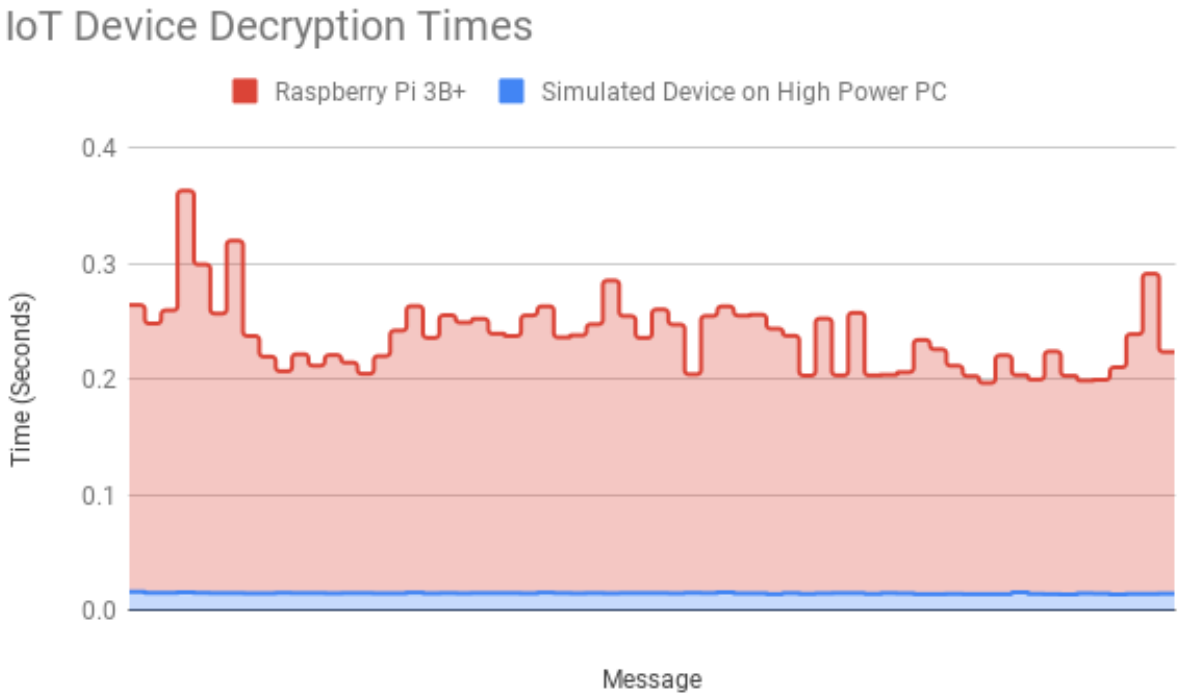

Figure 1: Displays decryption times of different devices over 64 messages to each device.

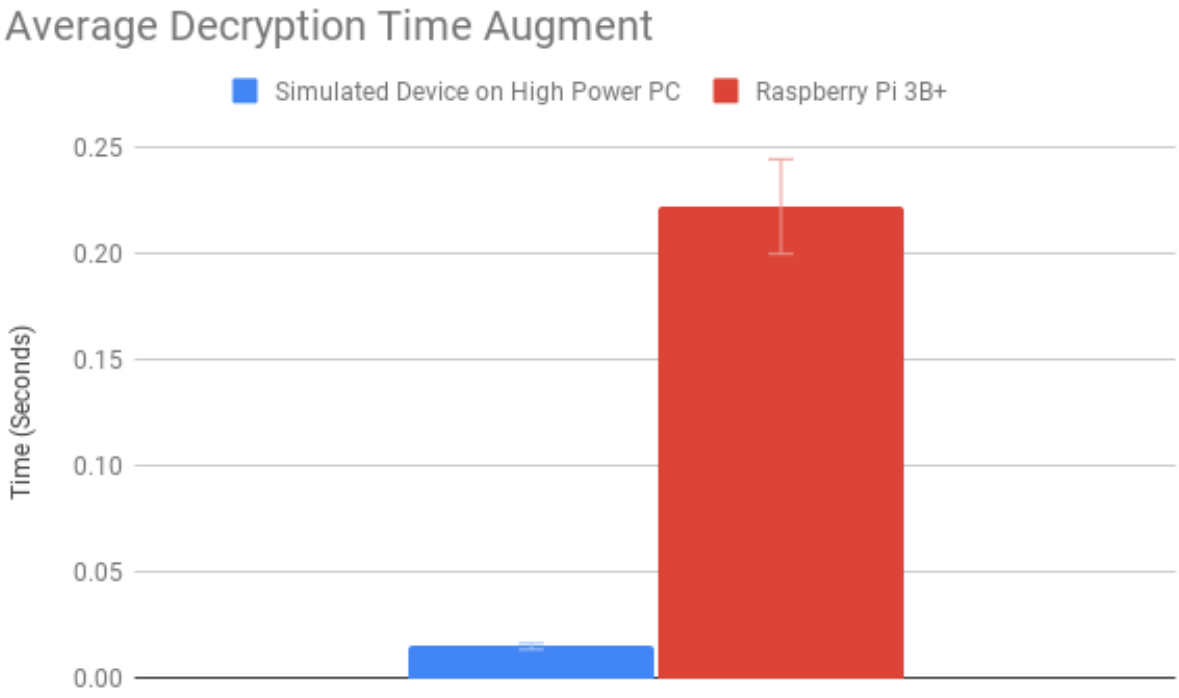

Figure 2: Visually represents average decryption times of different devices side-by-side.

As shown above, the Raspberry Pi's data was very noisy. Many factors may have led to this. The first is a lack of dedicated encryption hardware. Often, in a high end PC, there will

be integrated encryption logic boards on the motherboard for specific functions such as standard decryption operations. The Raspberry Pi Model 3 B+ is a low end device. Decryption times were, as expected, significantly slower on the Raspberry Pi than on the PC. The average augmentation to decryption time was around 0.22 seconds for the Pi, and under 0.02 seconds for the PC – a whole order of magnitude less. Both times are still unacceptable for standard file transport, and the Pi's time is unacceptable for even simple text/JSON messages.

In addition, this method will increase local network bloat. A standard exchange of single encrypted message will take exactly one message from the router to the intended recipient. However, with this method, there are 4 messages for every IoT device. The first is the message entering the network from the router to the hub (Step 5). The second is from the hub to the IoT device (Step 6). The third is from the IoT device to the hub (Step 8). Finally, the hub gives the message to the intended recipient (Step 9). This implementation will significantly burden low-end networks, which may not have the capacity to handle such a substantial increase in traffic. For our network with n=2, we had 8 times higher network traffic.

There is room for improvement via batching: this bloat could be reduced to $2n$+2 with the batching of the initial and final messages. Firstly, the smarthub would receive every encrypted chunk in a single message (Step 5, modified). Second, it would distribute the chunks as necessary (Step 6). Third, it would receive them back from each IoT (Step 8). However, the hub would wait for the messages now, instead of the client (Move step 10 to the smart hub). Finally, the hub would send a batch to the client (Step 9, modified).

4 Conclusion and Future Work

This solution is not feasible to implement. Latency is too high, and the weakness of a compromised smarthub is non-negligible, where an attacker with the right keys could still act as a man-in-the middle. Although this exact solution is not practical to implement in the real world, it lays a strong theoretical groundwork for other distributed cryptographic applications to build on.

If we were to continue working on this implementation, we could add to the protocols, such as authentication. This would allow us to truly identify users and IoT devices in the network. Aside from adding to the implementation, research could be done regarding public key length and its impact on decryption times for IoT devices. For our implementation we used the key length of 2048. Testing could be done with bigger and smaller key lengths to see the correlation with decryption times. Another research topic to look into would be using symmetric-key cryptography instead of public key cryptography for the implementation, specifically looking into the affects of symmetric-key cryptography on the decryption times.

References

[1] Agrawal, S., Mohassel, P., Mukherjee, P. and Rindal, P., (2019). DiSE: Distributed Symmetric-key Encryption. Csrc.nist.gov. Retrieved November 28, 2021, from https://csrc.nist.gov/CSRC/media/Events/NTCW19/papers/paper-AMMR.pdf.

[2] Andrychowicz, M. and Dziembowski, S., (2014). Distributed Cryptography Based on the Proofs of Work. Eprint.iacr.org. Retrieved October 9, 2021, from https://eprint.iacr.org/2014/796.pdf.

[3] Barrett, I., & Todorov, L. Distributed Cryptographic IoT Network [Computer software]. Retrieved December 2, 2021, from https://github.com/hornetfighter515/dist-iot-net

[4] Cachin, C., (2013). Distributed Cryptography. Cachin.com. Retrieved October 9, 2021, from https://cachin.com/cc/sft13/distcrypto.pdf.

[5] Henriques, M. S., & Vernekar, N. K. (2017, October 17). Using symmetric and asymmetric cryptography to secure communication between devices in IOT. IEEE Xplore. Retrieved November 28, 2021, from https://ieeexplore.ieee.org/document/8073643.

[6] Liris, & al. "PyPI Websocket Client Library." Python Package Index, Python Software Foundation. Retrieved 2 December 2021, from https://pypi.org/project/websocket-client/.

[7] Microsoft Research. (2018). Homomorphic Encryption - Microsoft Research. Retrieved October 9, 2021, from https://www.microsoft.com/en-us/research/project/homomorphic-encryption/.

[8] Pithikos, & al. "Python Websocket Server." Github, Microsoft. Retrieved 2 December 2021, from https://github.com/Pithikos/python-websocket-server.

[9] Shamir, A. (1979). How to Share a Secret. Web.mit.edu. Retrieved October 9, 2021, from http://web.mit.edu/6.857/OldStuff/Fall03/ref/Shamir-HowToShareASecret.pdf.

[10] Stüvel, S., & al, E. (n.d.). "PyPI RSA Library." Python Package Index, Python Software Foundation, 26 Nov. Retrieved December 2, 2021, from https://pypi.org/project/rsa/.

[11] Szerwinski, R., & Güneysu, T. (2008). Exploiting the power of gpus for asymmetric ... - springer. Exploiting the Power of GPUs for Asymmetric Cryptography. Retrieved November 28, 2021, from https://link.springer.com/chapter/10.1007%2F978-3-540-85053-3_6.

[12] Zhang, Z.-K., Cheng Yi Cho, M., Wang, C.-W., Hsu, C.-W., Chen, C.-K., & Shieh, S. (2014, December 8). IOT security: Ongoing challenges and research opportunities. IEEE Xplore. Retrieved December 1, 2021, from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6978614.