A Novel Approach Toward the Prevention of the Side Channel Attacks for Enhancing the Network Security

Suchismita Gupta
ProphecySensorlytics India
Bikramjit Sarkar
JIS College of Engineering
Subhrajyoti Saha
Texas Instruments (India)
Indranath Sarkar
JIS College of Engineering
Prasun Chakrabarti
ITM SLS Baroda University
Sudipta Sahana
University of Engineering and Management
Tulika Chakrabarti
Sir Padampat Singhania University
Ahmed A. Elngar (elngar_7@yahoo.co.uk)
Beni-Suef University

Research Article

Keywords: Side-Channel attack, timing attack, cache attack, shared library

Posted Date: September 20th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2074983/v1

Abstract

Privacy protection is an essential section of information security. The use of shared resources demands more privacy and security protection, especially in cloud computing environments. The aim of side-channel attacks is to extract secrets from systems. This can be through measurement and analysis of physical parameters. Execution time, electromagnetic emission, and supply current are some examples of such parameters. A side-channel attack does not target a program or its code directly. Instead, a side-channel attack attempts to gather information or influence the program execution of a system by measuring or exploiting the indirect effects of the system or its hardware. Put simply, a side-channel attack breaks cryptography by exploiting information inadvertently leaked by a system. The modules with integrated cryptographic systems pose a serious threat to these attacks. It has been observed that many robust algorithmic cryptographic operations have been broken successfully by side-channel analysis techniques. In this paper, the aim is to present a comparative review of the various side channel attacks possible and their countermeasures. Also, a new approach is proposed to prevent side-channel attacks and enhance the security of the entire network. The primary objective of this survey is to provide researchers in the field of side-channel attack a comprehensive summary of the progress achieved so far and to facilitate them to identify a few challenging future research areas.

Introduction

The process of taking preventative measures to protect the network that is underlying the infrastructure from unauthorised access, malfunction, destruction, improper disclosure, misuse or modification is defined as network security by SANS Institute. Implementing these measures allows computers, users and programs to perform their permitted critical functions within a secure environment. Securing a network requires a complex combination of hardware devices, such as routers, firewalls, and anti-malware software applications. Government agencies and businesses employ highly skilled information security analysts to implement security plans and constantly monitor the efficacy of these plans.

Computers and other devices connected to unsecured networks are highly vulnerable to external threats such as malware, ransomware, and spyware attacks. A single attack can bring down the entire computer system of an organization and compromise our personal information. By assuring the security of the network – typically with the assistance of a network security specialist – we can stay away from such expensive threats.

No matter whether we are an organization or an individual, our identity is valuable. If we log into an unsecured network, our identity can become visible to third parties. To avoid such a situation, we should secure our network. Such an approach becomes mandatory if we are a business that deals with client information.

When it comes to a business, special precautions should be taken to protect shared data. And, network security is one of the best ways to do so. Network security can be applied with different restrictions on

different computers depending on the types of files they handle.

In an unrestricted, unprotected network, network activity can become too heavy. Intense traffic can lead to an unstable computer network. Eventually, the complete network will become vulnerable to various external attacks.

Apart from the direct threat to the data in terms of security there lies another threat that is indirect and is termed as Side-channel attacks (SCAs). A side-channel attack does not target a program or its code directly. Instead, a side-channel attack attempts to gather information or influence the program execution of a system by measuring or exploiting indirect effects of the system or its hardware. A side-channel attack breaks cryptography by exploiting information inadvertently leaked by a system.

Although side-channel attacks are historically difficult to do, they are now more common because of several factors. The increasing sensitivity of measuring equipment has made it possible to gather extremely detailed data about a system while it is running. In addition, greater computing power and machine learning enable attackers to better understand the raw data they extract. This deeper understanding of targeted systems enables attackers to better exploit subtle changes in a system. Attackers can also go after high-value targets, such as secure processors, Trusted Platform Module (TPM) chips, and cryptographic keys. Even having only partial information can assist a traditional attack vector, such as a brute-force attack, to have a greater chance of success. Side-channel attacks can be tricky to defend against. They are difficult to detect in action, often do not leave any trace, and may not alter a system while it's running. Side-channel attacks can even prove effective against air-gapped systems that have been physically segregated from other computers or networks. Additionally, they may also be used against virtual machines (VMs) and in cloud computing environments where an attacker and target share the same physical hardware. Side Channel Attacks can be broadly classified into various types mentioned below:

Electromagnetic: An attacker measures the electromagnetic radiation, or radio waves, given off by a target device to reconstruct the internal signals of that device. The earliest side-channel attacks were electromagnetic. van Eck phreaking and the National Security Agency’s (NSA) Tempest system could reconstruct the entirety of a computer's screen. Attackers focus modern side-channel attacks on measuring the cryptographic operations of a system to try and derive secret keys. Software-defined radio (SDR) devices have lowered the barrier of entry for electromagnetic attackers, which can be performed through walls and without any contact with the target device.

Power: A hacker measures or influences the power consumption of a device or subsystem. By monitoring the amount and timing of power used by a system or one of its subcomponents, an attacker can infer the activity of that system. Some attacks may cut or lower power to cause a system to behave in a way beneficial to the attacker, similar to Plundervolt attacks.

Timing: Analyses the time a system spends executing cryptographic algorithms. A bad actor uses the length of time an operation takes to gain information. The total time can provide data about the state of

a system or the type of process it is running.

Here, the attacker can compare the length of time of a known system to the victim system to make accurate predictions.

Memory Cache: An attacker abuses memory caching to gain additional access. Modern systems use data caching and pre-fetching to improve performance. An attacker can abuse these systems to access information that should be blocked. The Spectre and Meltdown vulnerabilities that primarily affected Intel processors exploited this channel.

Acoustic: The attacker measures the sounds produced by a device. Proof-of-concept (POC) attacks have been performed that can reconstruct a user's keystrokes from an audio recording of the user typing. Hackers can obtain some information by listening to the sounds emitted by electronic components as well.

Optical: An attacker uses visual cues to gain information about a system. Although rarely used against computers, some POC attacks have been performed where audio can be reconstructed from a video recording of an object vibrating in relation to sounds. Simple shoulder surfing attacks may also fall into this category.

Hardware Weakness: Hackers can use physical characteristics of a system to induce behaviour, cause a fault or exploit data remanence, which is data that persists after deletion. Row hammering attacks happen when an attacker causes a change in a restricted area of memory by quickly flipping, or hammering, another area of memory located close by on the physical random-access memory (RAM) chip. Error correction code (ECC) memory can help prevent this attack. In a cold boot attack, the attacker quickly lowers the temperature of RAM, causing some of the information to be retained after power is removed so the attacker can read it back.

Organizations can implement a few best practice mitigations that may be good countermeasures for side-channel attacks. These attacks usually require specific detailed knowledge of a system to execute; therefore, a business should keep details related to implementation and vendors as a trade secret.

Address space layout randomization (ASLR) can prevent some memory- or cache-based attacks. Using business-grade equipment can also help to prevent systems from being exploited. Physical access to systems should be restricted as well. Businesses can also keep sensitive systems in shielded Faraday cages, and power conditioning equipment can shield against power attacks.

As extreme mitigations, increasing the amount of noise in a system will make it more difficult for an attacker to gain useful information. This paper provides a brief study of the various side-channel attacks. Further, it aims at proposing a way to prevent the side channel attack and thus, enhance network security.

Proposed Work

Let, P1 and P2 are two processes running on the same CPU core. Both are using some function from a shared library. So, the physical memory frame of the function Fs is shared by both P1 and P2.

Now let's say P1 starts executing. It fetches a lot of garbage instructions from the memory which replaced all the cache lines with garbage. At this point, P1 is certain that Fs are not in the cache.

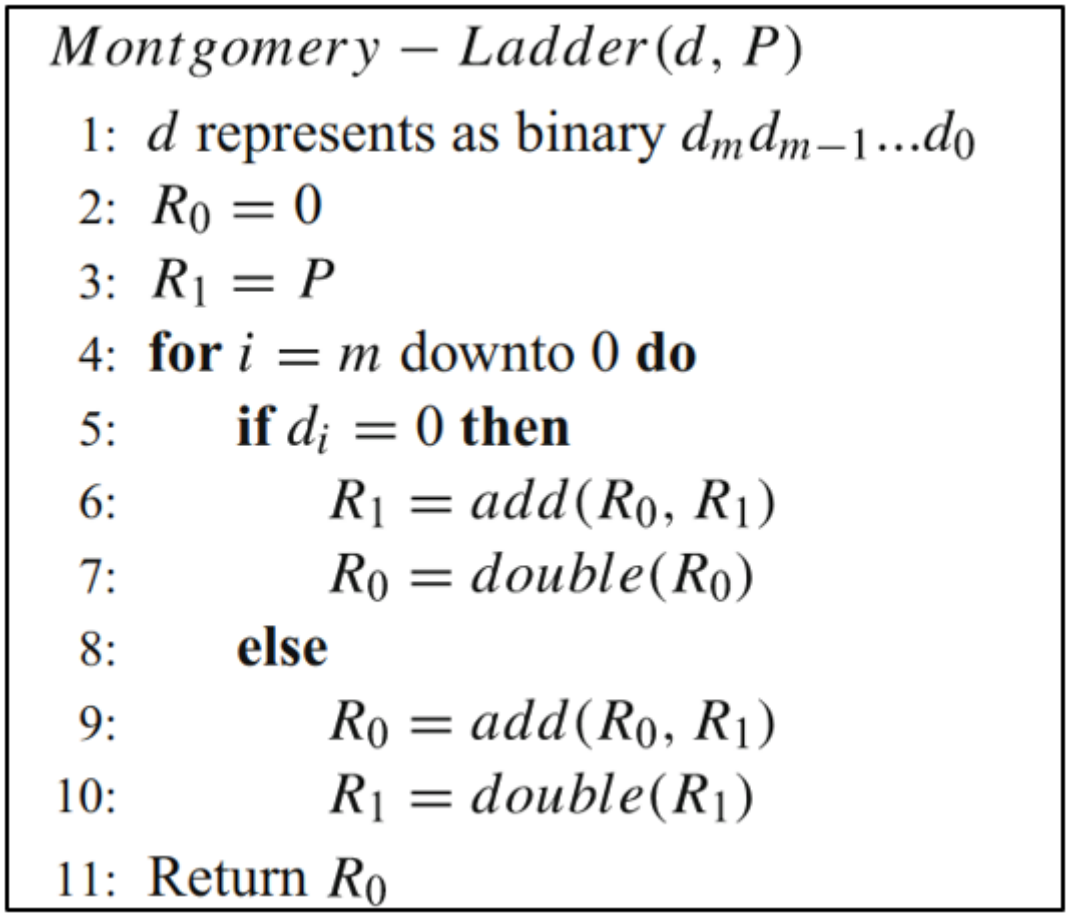
Now it's time for P2 to execute. If P2 needs Fs it will face a cache miss. So, the frame Fs will be brought back to the cache for execution. After executing for some time context switch happens again.

Now P1 comes online. This time P1 tries to fetch Fs and monitors the timing. If P2 has used Fs in the previous cycle, then P1 will have a cache hit, taking a lower time to execute instructions in Fs. Otherwise, P1 will face a cache miss, which takes relatively longer. So by analysing the timing information, P1 can determine if P2 has used Fs or not.

This is a kind of cache-side channel attack. The side channel here is time. There are small variations in this information retrieval algorithm, but the gist is the same.

Now it is not always a problem. It does not matter if another process knows my process is using the rendering library, or some maths library etc. It can't do any harm or alter the execution of my process from that information. But it becomes a serious issue when someone can recover secret cryptographic keys using this technique. Let us see an example,

Recovering cryptographic key-



The above figure (1) is an example of a RSA algorithm implementation. The d_i 's are digits of the secret key that must be private to the program executing this algorithm. These kinds of algorithms are widely used by millions of programs. So, generally OS's have a shared library containing the instructions of the algorithm. That way all the programs using this do not need to have a copy of the instructions, thus reducing memory usage of the overall system.

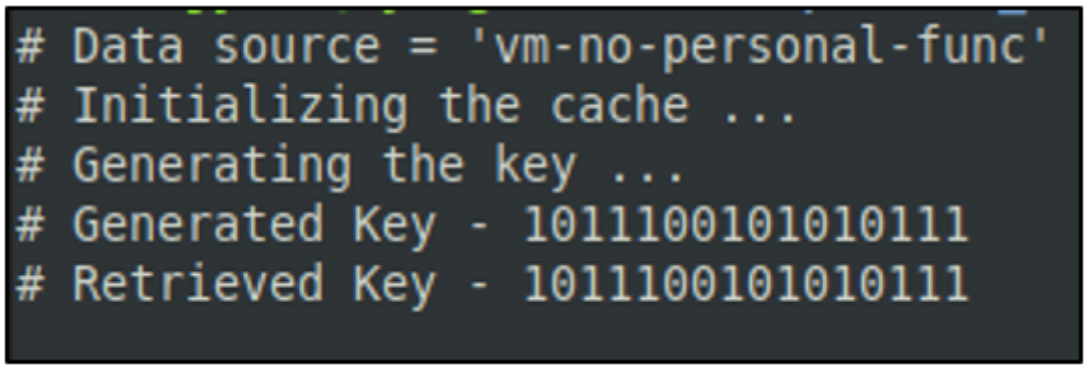
Here the above figure (1) is our SL. Suppose, P1 is executing the algorithm using it's own secret d_i's. Let's see how P2 can recover values of d_i's -

- P1 executes for some time.
- Context switch.
- P2 flushes the cache or loads some random useless data to the cache so that the cache does not contain memory blocks corresponding to the line numbers 6 & 7 of figure 1.
- P2 starts waiting for some time - context switch.
- P1 starts executing. Now here are two possibilities.
    - If the current value of d_i is '0' then P1 will fetch the memory blocks of lines 6 & 7 from RAM to cache and execute.

    - If the current value of d_i is '1' then P1 will execute lines 9 & 10. So, lines 6 & 7 are still absent in the cache.
- Context switch happens.
- P2 loads the instructions on lines 6 & 7.
    - If the instructions run relatively fast (faster from a predefined threshold), then it can be said that it is a cache hit. So, in the previous cycle, P1 must have used those instructions. So, the value of d_i in the previous cycle was '0'.
    - If the instructions run relatively slow, then it's a cache miss. So, in the previous cycle P1 did not use those instructions. So, d_i was '1'.

Following this above procedure, P2 can get the secret key of P1.

We have simulated this attacking process virtually in a c program. Here one function (simulating victim process, vctm_proc) generates a 16-bit key in its personal block. The key is never passed to anyone. Despite that, another function (simulating the attacker, atkr_proc) is able to retrieve the key from timing analysis. The output is something like this,



In practical computing environments, due to background noises (multi-core CPU, multi-level cache, other running processes, and OS itself), the execution of timing-based side-channel attacks is more complicated than that. The attacker needs to analyze timing data statistically over many execution cycles. But the basic principle is the same.

Here we have taken the situation most suitable for the attacker. So the attacker can retrieve the key in only one attempt. Our argument is that, if we can prevent attackers from retrieving keys in the most favorable situation, the key is safe in all other possible scenarios.

So how do we do it?

Let's first analyze where the problem begins. If the execution of a block of code in the shared library does not depend on the value of the key, then the attacker will not be able to gain any extra information other than the victim process using that library, which is not a problem here. But when there is a conditional

execution of a block of code in the shared library whose condition depends on the value of the key, then the attacker is able to gain extra information.

Here two terms are important, key dependent block, and key independent block.

Now our solution is simply “avoid sharing the key dependent block between processes.” Provide copies of the dependent block in separate physical locations for each process.

In static libraries, the libraries are copied to the programs using them in compile time which takes a lot of memory. On the other hand, shared libraries pose the aforementioned security loopholes in some cases. So we have chosen a middle round.

Programmer’s perspective-

It is kept in the hand of the programmers of the shared library to decide which parts/functions of the library may reveal secrets and which parts are okay when shared. The parts that should not be shared are marked using a keyword, say, “personal” before the function name. It will look like this,

```
1.   d represents as binary d_m d_{m-1} ... d_0
2.   R_0 = 0
3.   R_1 = P
4.   for i = m downto 0 do
5.       encrypt ( d_i, R_0, R_1 )
6.   Return R_0
7.   personal function encrypt ( d_i, R_0, R_1 ) {
8.       if d_i = 0 then
9.           R_1 = add ( R_0, R_1 )
10.          R_0 = double ( R_0 )
11.      else
12.          R_0 = add ( R_0, R_1 )
13.          R_1 = double ( R_1 )
14.  }
```

Now the programs that uses this library does not get any copy of the library code at compilation time. So, unless the program is executed, there is no memory overhead.

When a program is loaded into the memory for execution, during that time, the instructions in the personal functions in the shared library have to be copied in a separate memory block and mapped to the virtual memory of the process by manipulating the page table of the process. This job has to be done by the loader with the help of the kernel.

A similar simulation using the personal function method shows the effectiveness of this technique,

```
# Data source = 'vm-personal-func'
# Initializing the cache ...
# Generating the key ...
# Generated Key - 1011100101010111
# Retrieved Key - 1111111111111111
```

Memory overhead-

Let a shared library has N instructions, of them m portion is defined as personal functions. So total number of shared instructions is, N*m

Let there be P programs using a shared library in a computer. Among those, the p portion is currently being executed. So the total number of copies of those personal functions is, P * p

So total memory being used by the library is,

$P \cdot p \cdot N \cdot m + N \cdot (1 - m)$ instruction length

For comparison purposes, using only static libraries will take up space,

Ptot * N instruction length (put m = 1 and p = 1 in prev expression)

And for using only a shared library without personal functions will take,

N instruction length (put p = 0 and m = 0 in the prev expression)

Here is how they compare graphically,

Here X axis is P and Y axis is memory used.

The Red is using static library,

Blue is using shared library, without personal functions.

Green is using shared libraries with personal functions.

Now, if there are total n libraries in a system, we have memory consumption

Σ Pi * pi * Ni * mi + Ni * (1 - mi), for all i = 1:1:n

Implementation Method

The job of implementing personal functions must be handled by the dynamic linking loader. During the compilation of the shared object from the source to object file, all the personal functions should be under a separate section that exists in a physically separate block (minimum allocatable memory in RAM) of memory. When the library is called by a process, the loader will load the shared code of the library as well as the personal functions to the ram. If the library is already in the ram, then only the personal functions are copied. Then the codes are linked.

But here is a catch, how the shared part of the code will be able to call the same personal functions from different locations depending on the process. To achieve this, we can use page tables. While execution, the relative addresses of the whole library are the same for all the processes using that library. For the shared part, the virtual address will point to the same physical address but for personal functions, this will point to different physical memory locations for different processes. For this reason, the shared and the personal part of the library must lie in separate pages in the virtual memory space as well as separate frames in the physical memory space. All of these can be done by the dynamic linking loader.

Conclusion

Keeping the need for network security in mind this paper provides a well-explained study of one of the major concerning fields for any network security person, i.e., side-channel attack. This paper presents a countermeasure of the Cache Side channel attack by proposing a comprehensive way of designing the cache memory usage. This would ultimately serve the purpose of security. The system model that is being used in this paper, is considered ideal for the attacker to steal data. Our vision was to make secret data leak through timing-based cache side-channel attack theoretically impossible. If one cannot steal data from the most suitable system for the side channel attack, the attack will evidently fail in more complex real-world cases. Following this perspective, we will be able to make the shared libraries ecosystem more robust and secure without compromising memory efficiency a lot.

Declarations

Human and Animal Ethics

Not Applicable

Acknowledgments

The corresponding author is thankful to the co-authors for their support to complete the manuscript. We are also thankful to the anonymous reviewers and Editors for their perceptive comments on the manuscript.

Ethical Approval

The manuscript is original and has not been published, accepted for publication or under editorial review for publication elsewhere.

Consent to Participate

Not applicable

Consent to Publish

Not applicable

Funding

No funding was received to assist with the preparation of this manuscript.

Availability of data and materials

All data generated or analyzed as part of this study are included in this published article.

Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests

There is **No known competing** financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. P. C. Kocher, “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems,” in Advances in Cryptology—CRYPTO (Lecture Notes in Computer Science), vol. 1109. Barbara, CA, USA: Springer, 1996, pp. 104–113.

2. Kocher, P.; Jaffe, J.; Jun, B. Differential power analysis. In Annual International Cryptology Conference; Springer: Berlin/Heidelberg, Germany, 1999; pp. 388–397
3. Bernstein, D.J. Cache-Timing Attacks on AES. 2005. Available online: https://cr.yp.to/antiforgery/cachetiming-20050414.pdf (accessed on 15 September 2021).
4. Fong, X.; Choday, S.H.; Roy, K. Design and optimization of spin-transfer torque mrams. In More than Moore Technologies for Next Generation Computer Design; Springer: Berlin/Heidelberg, Germany, 2015; pp. 49–72.
5. Gandolfi, K.; Mourtel, C.; Olivier, F. Electromagnetic analysis: Concrete results. In International Workshop on Cryptographic Hardware and Embedded Systems; Springer: Berlin/Heidelberg, Germany, 2001; pp. 251–261.
6. Brier, E.; Clavier, C.; Olivier, F. Correlation power analysis with a leakage model. In International Workshop on Cryptographic Hardware and Embedded Systems; Springer: Berlin/Heidelberg, Germany, 2004; pp. 16–29.
7. Debayan Das and Shreyas Sen, “Electromagnetic and Power Side-Channel Analysis: Advanced Attacks and Low-Overhead Generic Countermeasures through White-Box Approach”, Article of MDPI, Published: 31 October 2020.
8. Agrawal D., Archambeault B., Rao J.R., Rohatgi P. (2003) The EM Side—Channel(s). In: Kaliski B.S., Koç .K., Paar C. (eds) Cryptographic Hardware and Embedded Systems - CHES 2002. CHES 2002. Lecture Notes in Computer Science, vol 2523. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-36400-5_4.
9. Lu Zhang, Luis Vega, Michael Taylor, “Power Side Channels in Security ICs: Hardware Countermeasures”, arXiv:1605.00681v1 [cs.CR] 2 May 2016.
10. Debayan Das, Arijit Raychowdhury, Santosh Ghosh and Shreyas Sen, “EM/Power Side Channel Attack: White-Box Modeling and Signature Attenuation Countermeasures”, Digital Object Identifier 10.1109/MDAT.2021.3065189 Date of publication: 15 March 2021; date of current version: 20 May 2021.
11. Yangdi Lyu and Prabhat Mishra, “A Survey of Side-Channel Attacks on Caches and Countermeasures”, © Springer International Publishing AG, part of Springer Nature 2017.
12. D.Brmley and D.Bosh. “Remote timing attacks are practical”, in USENIX, August 2003.
13. Edward W. Felten and Michael A. Schneider. “Timing Attacks on Web Privacy”, Secure Internet Programming Laboratory, Univ. Princeton, Princeton, NJ 08544 USA
14. Janaka Alawatugoda, Roshan Ragel and Darshana Jayasinghe; "Countermeasures Against Bernstein's Remote Cache Timing Attack", in Proceedings of the 6th IEEE International Conference on Industrial and Information Systems (ICIIS2011), Kandy, Sri Lanka, August 2011.
15. Darshana Jayasinghe, Roshan Ragel and Dhammika Elkaduwe, “Constant Time Encryption as a Countermeasure against Remote Cache Timing Attacks”, in Proceedings of the 6th International Conference on Information and Automation for Sustainability (ICIAfS 12), Beijing, China, September 2012.

16. Eran Troman, Dag Arne Osvik and Adi Shamir, “Efficient Cache Attacks on AES, and Countermeasures”, Journal of Cryptiology, J.Cryptol.(2010) 23:37-71 DOI:10.1007/s00145-009-9049-y.
17. Backes, Michael et al. “Acoustic Side-Channel Attacks on Printers.” *USENIX Security Symposium* (2010).
18. Tzipora Halevi and Nitesh Saxena, “Keyboard acoustic side channel attacks: exploring realistic and security-sensitive scenarios”, © Springer-Verlag Berlin Heidelberg 2014, Int. J. Inf. Secur. DOI 10.1007/s10207-014-0264-7.
19. H.S. Wang, D.G. JI, Y Zhang, K.Y Chen, J.G. Chen, Y.Z. Wang, “Optical Side Channel Attacks on Singlechip”, 10.2991/itms-15.2015.87.
20. Ricardo Villanueva-Polanco, “Cold Boot Attacks on LUOV”, Appl. Sci. 2020, 10(12), 4106; https://doi.org/10.3390/app10124106.
21. R. Xu et al., "Side-Channel Attack on a Protected RFID Card," in IEEE Access, vol. 6, pp. 58395-58404, 2018, doi: 10.1109/ACCESS.2018.2870663.

Graph

Graph is available in the Supplementary Files section.

Figures

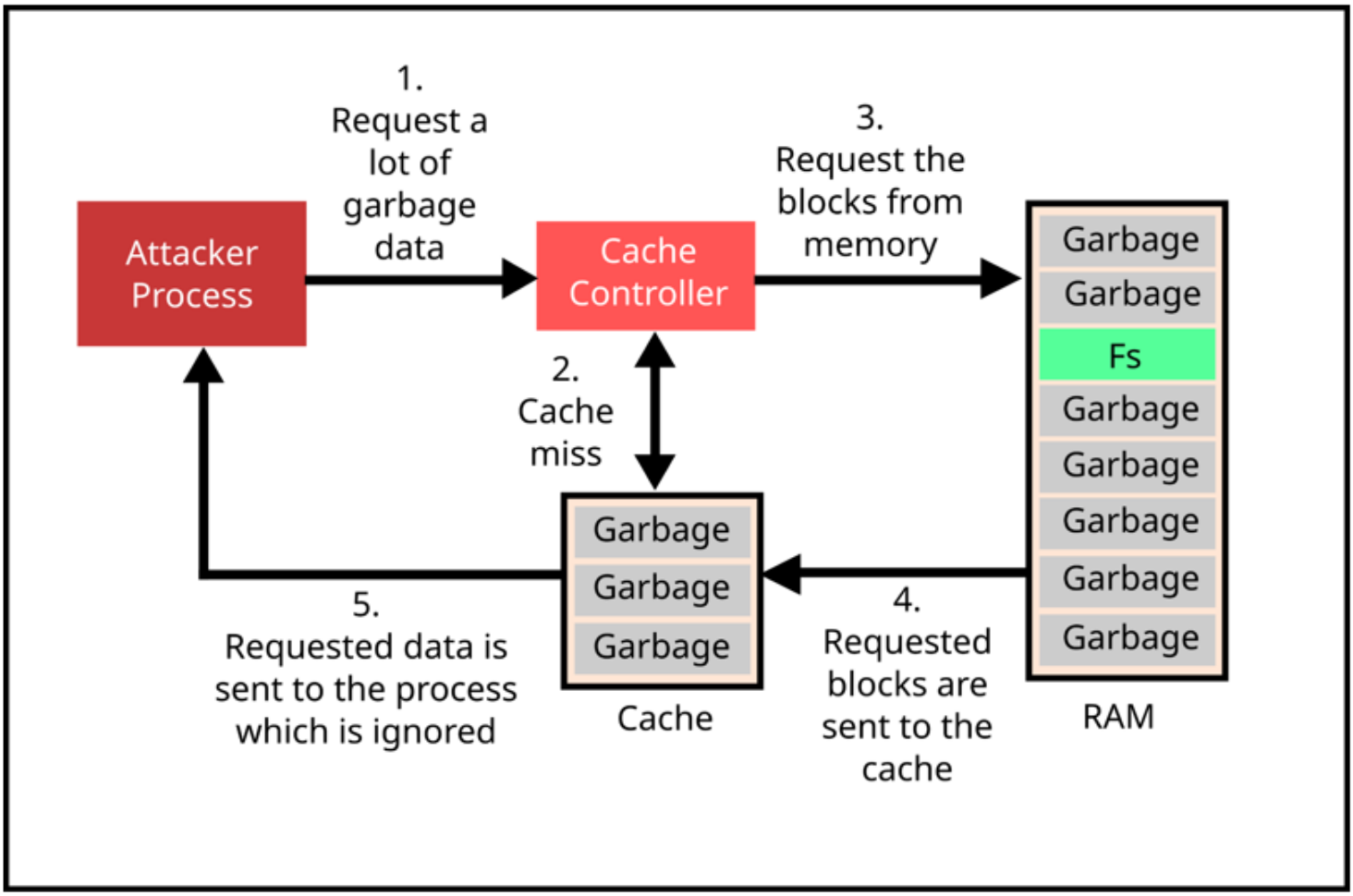


Figure 1

The Attacker (P1) request a lot of garbage data to make sure Fs is deleted from the Cache

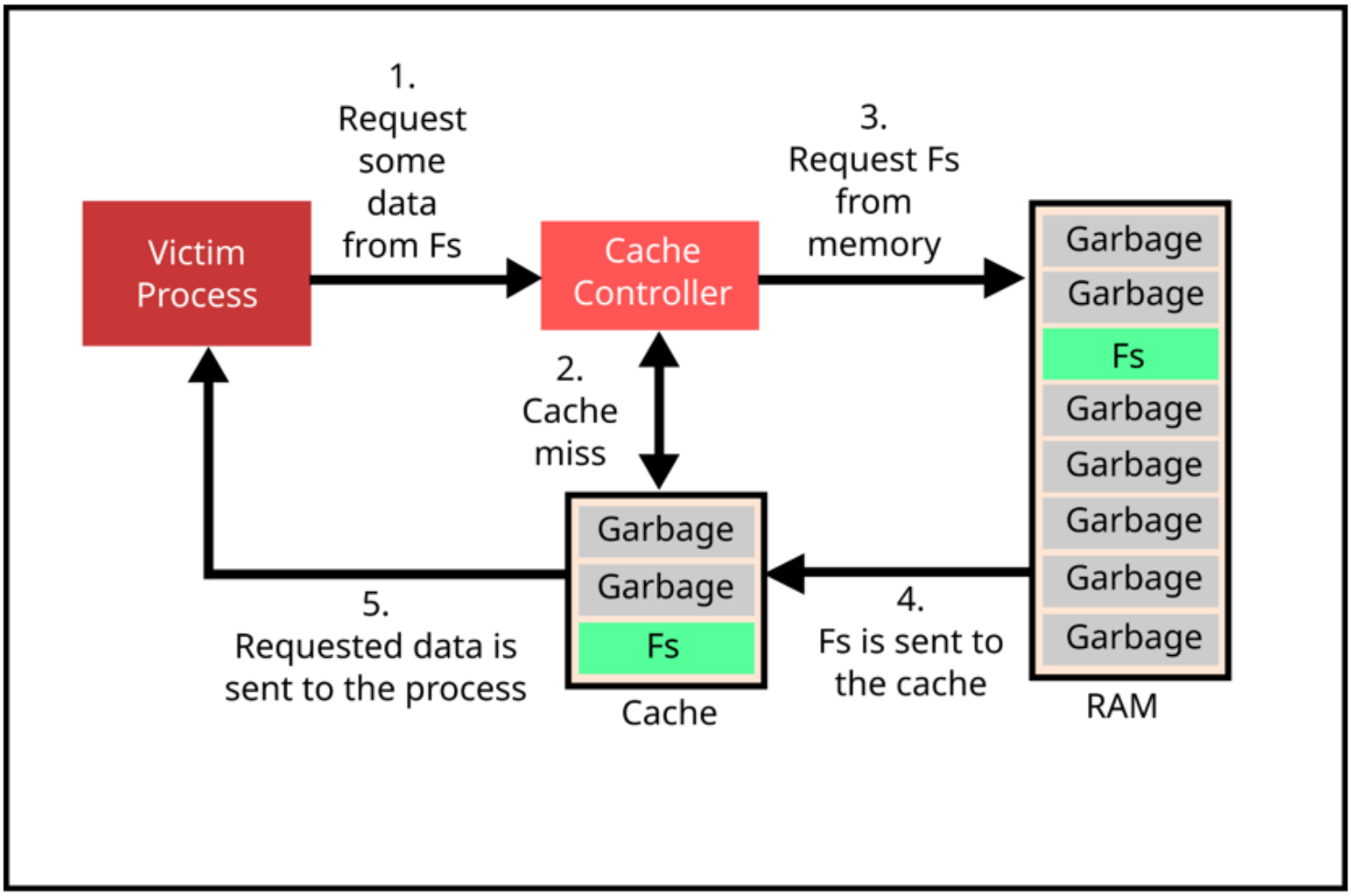


Figure 2

If the Victim (P2) used Fs it is again copied to the Cache

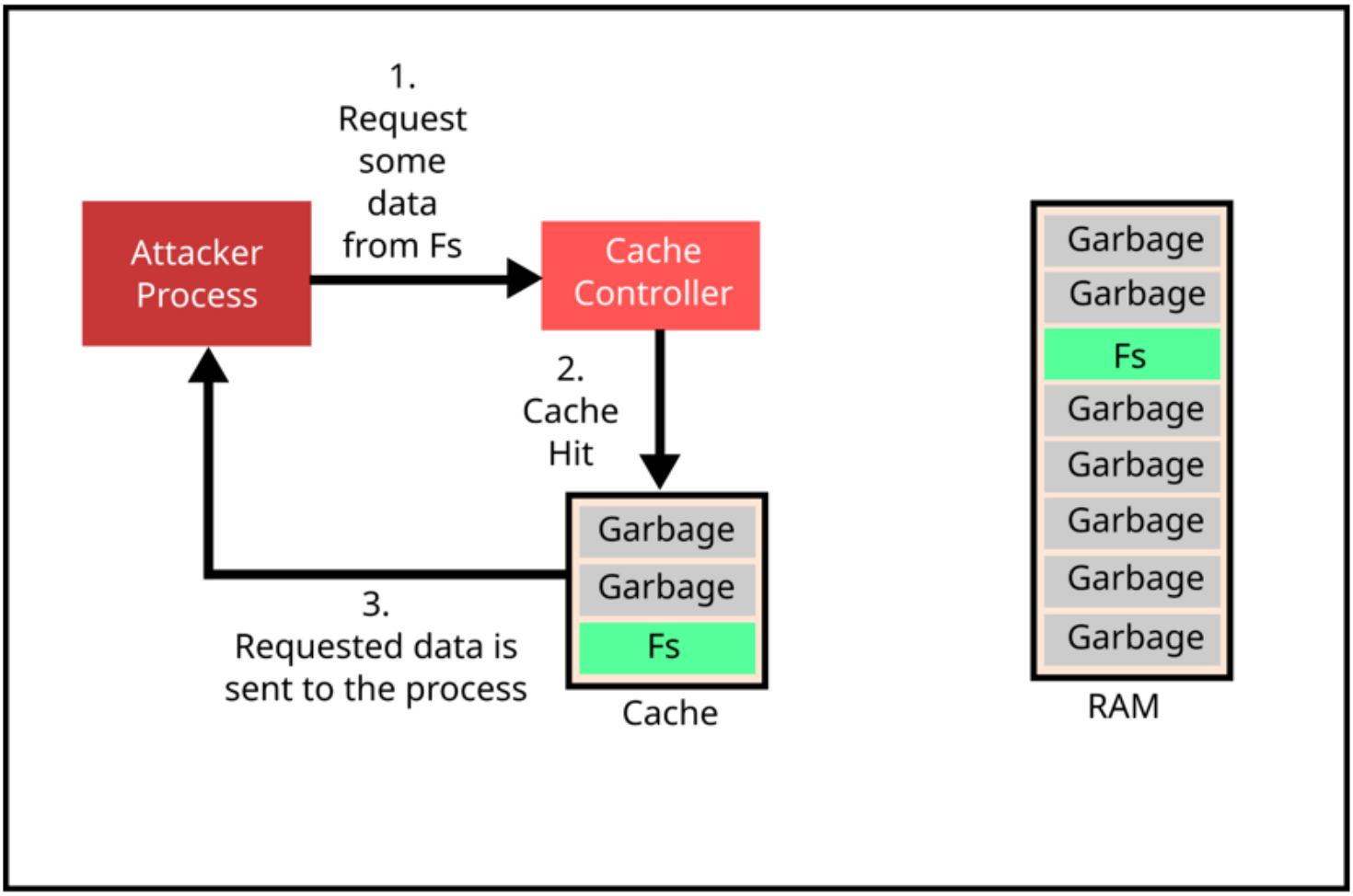


Figure 3

If P2 have used Fs, P1 will have cache hit in a request to Fs

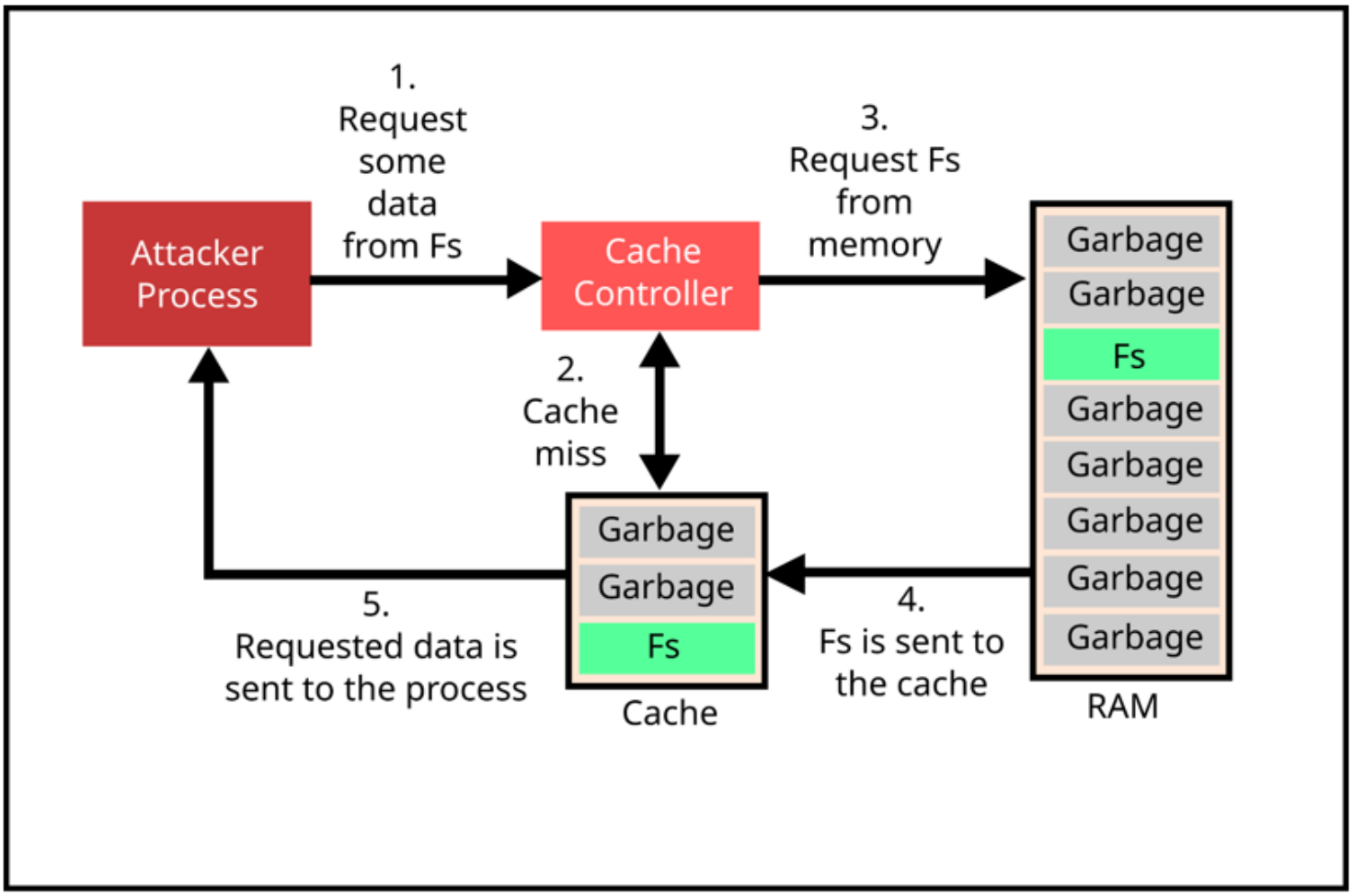


Figure 4

If P2 haven't used Fs, P1 will face a cache miss in a request to Fs

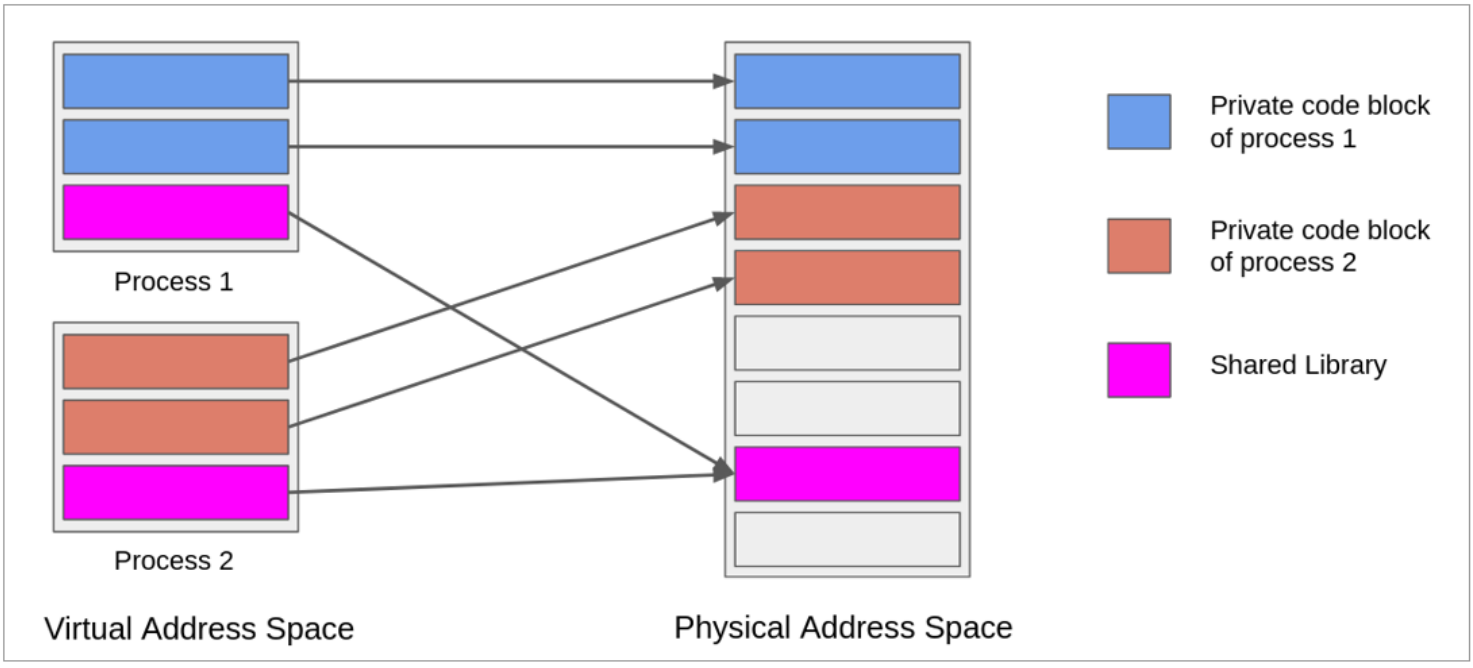


Figure 5

Memory model of a normal shared library

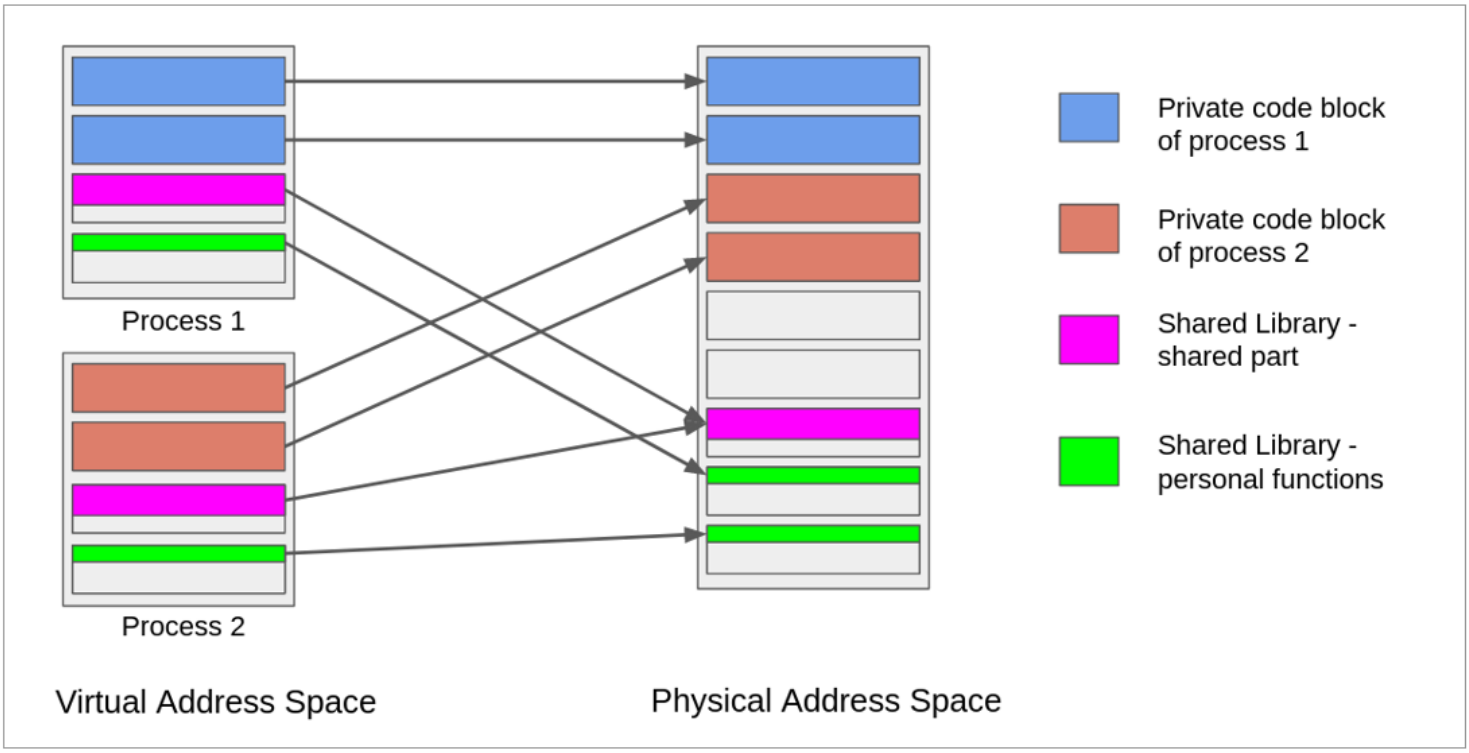


Figure 6

Memory model of a shared library with personal functions

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- Highlights1.docx
- grapgh.png