SIMurai: Slicing Through the Complexity of SIM Card Security Research

Tomasz Piotr Lisowski, *University of Birmingham;* Merlin Chlosta, *CISPA Helmholtz Center for Information Security;* Jinjin Wang and Marius Muench, *University of Birmingham*

https://www.usenix.org/conference/usenixsecurity24/presentation/lisowski

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

SIMURAI: Slicing Through the Complexity of SIM Card Security Research

Tomasz Piotr Lisowski [†], Merlin Chlosta[‡], Jinjin Wang[†], Marius Muench[†]

[†]*School of Computer Science, University of Birmingham*
[‡]*CISPA Helmholtz Center for Information Security*

Abstract

SIM cards are widely regarded as trusted entities within mobile networks. But what if they were not trustworthy? In this paper, we argue that malicious SIM cards are a realistic threat, and demonstrate that they can launch impactful attacks against mobile devices and their basebands.

We design and implement SIMURAI, a software platform for security-focused SIM exploration and experimentation. At its core, SIMURAI features a flexible software implementation of a SIM. In contrast to existing SIM research tooling that typically involves physical SIM *cards*, SIMURAI adds flexibility by enabling deliberate violation of application-level and transmission-level behavior—a valuable asset for further exploration of SIM features and attack capabilities.

We integrate the platform into common cellular security test beds, demonstrating that smartphones can successfully connect to mobile networks using our software SIM. Additionally, we integrate SIMURAI with emulated baseband firmwares and carry out a fuzzing campaign that leads to the discovery of two high-severity vulnerabilities on recent flagship smartphones. We also demonstrate how rogue carriers and attackers with physical access can trigger these vulnerabilities with ease, emphasizing the need to recognize hostile SIMs in cellular security threat models.

1 Introduction

Today's connected society relies on the uninterrupted availability and security of cellular networks, and their perpetually connected mobile devices. Regardless whether these end devices are feature-rich smartphones, low-power IoT devices, or highly reliable industrial components—they all employ Subscriber Identity Modules (SIMs)[1] for authenticating and connecting to mobile networks. Owing to their small size, perceived narrow feature-set, and apparent single purpose—authentication—SIMs are generally trusted to the point where they may start acting as the root of trust [5, 8, 21].

[1]We refer to the ubiquitous notion of a 'SIM card', collectively as 'SIMs'.

Conversely, in this paper, we highlight the prospect of attacker-controlled SIMs, an often overlooked yet realistic attack vector. For instance, consider tourists traveling abroad, and purchasing a local SIM card from an unfamiliar operator to avoid roaming charges. Whether they get a prepaid SIM card from an untrusted store, or an eSIM from a carrier they haven't heard of before; their phone is now attached to a small networked computer that has a direct connection to their phone's baseband.

Typically, mobile networks are feature-rich, covering diverse use cases and offering specific capabilities for niche subscriber groups. While some features may become obsolete, their implementations remain within the protocols and software of the network. Modern operating systems, both for desktop and mobile computers, made major leaps in compartmentalization, sandboxing, and permission management, enhancing the overall security of end devices. However, real-time operating systems driving cellular baseband firmware, that manage nearly all aspects of mobile networks, often lack these modern security features [38]. Nonetheless, mobile operating systems, such as *Android* or *iOS*, prevent most apps and outside entities from directly interfacing with the baseband. SIMs, however, have a direct, privileged, and unfiltered interface to the baseband. Hence, we explore baseband security from the perspective of hostile SIMs. Contrary to popular belief, SIMs do more than just store keys; authentication is only one of their many offered facilities. Among other capabilities, they can also run Java applets, send and receive Short Message Service (SMS) messages, or open TCP/IP sockets.

Until now, research and testing with SIMs often relied on self-provisioned hardware SIMs, or rudimentary software stubs, with the sole task of authenticating to the network. While SIM contents can be programmed onto special physical SIMs, the card's operating system behavior is locked-down to be inflexible and standards-compliant, which limits the freedom of experimentation. Software SIMs can fill that gap, however, until now, there was no software SIM that would act as an analog to a standards-compliant SIM by default, while having the ability to violate that behavior when desired.

We close that gap by designing and implementing SIMURAI, a software research platform specializing in SIM card security exploration. Its main components are an open-source software SIM—swSIM—powered by the underlying smart card framework—swICC. These software components are standards-compliant by default but do not enforce this behavior, which provides the flexibility required for them to work as a research platform.

We evaluate SIMURAI in a series of experiments in common physical 2G/4G/5G cellular test beds, and a virtualized 4G setup, showcasing that SIMURAI is compatible with COTS smartphones and provides a viable alternative to hardware SIM cards in the lab. In addition, we also demonstrate SIMURAI's capabilities for security research: First, by replicating a SIM-based spyware. Then, using SIMURAI in combination with the FirmWire emulation platform [22] to enable a fuzzing campaign against emulated User Equipment (UE) images, we discover two high-severity memory corruptions that were acknowledged by the affected vendor. Motivated by the vulnerabilities found in baseband implementations, we evaluate the practicality of SIM-originating attacks against baseband implementations with two experiments: First, we analyze a *SIM interposer* and patch its firmware to launch an attack, illustrating that SIM attacks are feasible with short *physical access*. Second, we update the SIM card remotely to install a malicious applet, showing that *hostile SIMs* are a relevant threat to baseband security.

In summary, we make the following contributions:

- We stress the urgency to begin considering hostile SIMs as an attack vector, and the resulting need for security-focused SIM research tooling.
- We design and implement SIMURAI; an extensible software research platform for SIM security research, consisting of two main components: swSIM, a software SIM, and swICC, the underlying smart card layer.
- We demonstrate two case studies in which we use SIMURAI for security research. In particular, we implement a fuzzing campaign against commercial baseband firmware implementations, leading to the discovery of high-severity security vulnerabilities.
- We demonstrate that our discovered vulnerabilities can be exploited either through *(i)* physical access or *(ii)* a hostile SIM, and outline potential mitigations against SIM-based attacks.

We provide the full source code and research artifacts for SIMURAI at https://github.com/tomasz-lisowski/simurai.

2 Background

2.1 Cellular Baseband

Smartphones typically employ specialized processors for various tasks. The *baseband processor* handles mobile communication in its entirety. Its firmware is considered particularly complex by functionality and size, as it typically implements the full 2G, 3G, 4G and 5G network stacks, with the exception of the physical layer. This requires parsers for complex binary formats (e. g., ASN.1), an IP stack, an IP Multimedia Subsystem (IMS) stack, and more.

Across all these layers and protocols, the baseband firmware handles untrusted input. For instance, fake base stations may send SMS messages to nearby phones when using the 2G standard, or they might inject malicious 4G or 5G messages before mutual authentication occurs between the network and end devices.

Google notes that bugs in baseband implementations "may potentially be abundant," and calls for the adoption of additional security measures such as using updated build toolchains, deploying sanitizers in critical code, and transitioning to memory-safe languages [38]. At the same time, despite recent advances, security testing of baseband firmware remains a challenging and ongoing area of research (e. g. [24, 25, 31, 32, 34, 52]).

2.2 Subscriber Identity Module

Smart cards, i. e., SIM cards, bank cards, biometric passports, etc, are often built on top of an Integrated Circuit Card (ICC), extended with domain-specific augmentations to perform specialized tasks. ICCs contain a CPU, ROM, RAM, a file system, and a Card Operating System (COS) to manage resources and enforce hardware-assisted security mechanisms.

SIMs implement technologies in a layered fashion, which reflects the layering of standards used to define them. ETSI standardizes the UICC[2], which is a hardware and software architecture based on the ICC and intended for use in telecommunication applications such as SIM cards [13]. In the context of 3G, 4G, and 5G, 3GPP-issued standards extend the ETSI documents for SIMs operating on 3GPP networks. SIMs supporting the older, second-generation mobile telecommunication technology (2G), must instead implement GSM standards maintained by ETSI and 3GPP.

Communication Protocol. All communication between terminals[3] and SIMs aims to exchange Application Protocol Data Units (APDUs). Table 1 shows the structure of APDUs, highlighting that each APDU is divided into two parts: *(i)* The

[2] UICC is not an acronym [13], although sometimes incorrectly referred to as a Universal ICC; similar to Universal SIM (USIM).

[3] In cellular networks, end devices are commonly referred to as *UE*, or *terminal* in the context of SIM communication.

Table 1: Structure of short command and response APDUs.

Command		
Field	**Description**	**Size in Bytes**
CLA	Command Class	1
INS	Instruction	1
P1	Parameter 1	1
P2	Parameter 2	1
L_c	Encodes length of command data	0 or 1
	Command Data	$0 \leq N_c \leq 255$
L_e	Encodes length of response data	0 or 1

Response		
	Response Data	$0 \leq N_e \leq 256$
SW1	First byte of the status word	1
SW2	Second byte of the status word	1

Command APDU (C-APDU), a command that is sent from the terminal to the SIM, and *(ii)* the Response APDU (R-APDU), a response sent from the SIM back to the terminal. Together, a C-APDU and R-APDU form a Command/Response Pair (C-RP) which completes the APDU. Note that in this protocol, the SIM may never issue C-APDUs and the terminal may never issue R-APDUs [12, 27].

Proactive Commands. APDUs are always initiated by the terminal, but the SIM's optional proactive features enable it to issue *proactive* commands, which the terminal executes. Once the SIM creates a proactive command, the next 9000 status word in the R-APDU is overwritten with a special 91XX value, that informs the terminal that it can fetch a proactive command. When timely execution of proactive commands is important, the terminal may be requested to begin polling the SIM with periodic STATUS APDUs; polling may also be enabled by default [10, 12].

Proactive commands are different from APDUs, but they remain feature-rich; for instance, they allow the SIM to display a menu, open the web browser, send an SMS message, or obtain the terminal's location [3, 10]. The proactive features supported by a terminal are communicated to the SIM via the TERMINAL PROFILE command.

Consider the PROVIDE LOCAL INFORMATION command shown in Figure 1. This instruction allows the SIM card to obtain a variety of information from the terminal, including: Date, time, time zone, IMEI of terminal, charge state of the battery, network measurement results, or, as seen in this example, location information. Note that all proactive commands are encoded with ASN.1 BER-TLV rules [28]: BER-TLV byte strings describe Data Objects (DOs). Each DO contains a tag, length, and value; where the value contains as many bytes as indicated by the length field. The value of a DO can itself be a DO, a list of DOs, or a binary blob [12, 27].

```
0xD009 8103 00 26 00 8202 81 82
D009 Tag: Proactive command
     Length: 9
     Value:
├─8103 Tag: Command details
│      Length: 3
│      Value:
│   ├──00 Command number: 0
│   ├──26 Command type: PROVIDE LOCAL INFORMATION
│   └──00 Information: Location according to current NAA
└─8202 Tag: Device identities
       Length: 2
       Value:
    ├──81 Source: UICC
    └──82 Destination: Terminal
```

Figure 1: Example of a BER-TLV-encoded PROVIDE LOCAL INFORMATION proactive command. This command reads location information from the UE based on the current Network Access Application (NAA), i. e., USIM in most SIM cases.

File System. Files are organized into a forest of trees, with Application DFs (ADFs) and one Master File (MF) at the root, each sitting at the root of their respective trees, with Dedicated Files (DFs) and Elementary Files (EFs) nested in each of them. DFs are similar to directories and EFs are comparable to files. The MF is a root of the whole forest, while also being a root of one of the trees. This is important as it hints at the older structure of the File System (FS), where ADFs did not exist, and the MF contained all files.

eSIM. An embedded SIM (eSIM) is based on the embedded UICC (eUICC), that is intended to be a non-removable replacement for the UICC with identical core functionality. This comprises applications, file system, and all other features described in Section 2.2, which remain present in the eUICC and work in largely the same manner. The core difference is that subscriber credentials—packaged into a 'profile'—need to be remotely provisioned onto the eUICC. Each profile is a representation of a file system and applications similar to those stored on a regular UICC. Installed profiles can be enabled and disabled, and they are isolated to avoid cross-profile communication [11].

Interposers. Interposers are paper-thin Machine-in-the-Middle (MITM) devices placed between SIMs and phones; they can be used to intercept and manipulate terminal-SIM communications (see Appendix B), and they are often sold to bypass SIM-locks, i. e., a lock that restricts the device to *exclusively* using the network of a specific operator.

SIM Research Tooling. Carrier-issued SIM cards can only be programmed by the carrier and thereby are not suitable for experimentation. As a result, suppliers like Sysmocom offer SIMs with administrative pins necessary for altering file contents and installation of smart card Java applets (cardlets) [63]. They can be programmed with any smart card reader and compatible software (e. g., *pySIM* [48] and *shadySIM* [35, 49] for cardlets). Additionally, specialized hardware, such as *SIMtrace2* [50], can function as a *SIM breakout board* to enable tracing of communication between SIMs and the terminals, or to provide an interface to the terminal's SIM reader.

3 Motivation

From a security perspective, SIM cards occupy a unique and crucial role in protecting UEs and ensuring user safety. They offer a comprehensive range of features and are widely regarded as a trustworthy component within the telecommunications stack. In this paper, we challenge this trust and make a case for considering *hostile SIMs* in modern telecommunication threat models.

Although, at first glance, controlling SIMs appears to put strong assumptions on the attacker, this capability can be achieved through various methods:

1. Physical access to a UE or SIM. Attackers can, with brief access, not only exchange the SIM card, but also install interposers manipulating the communication between a benign SIM and the baseband.

2. Remote SIM administration features. This is a standardized feature of SIMs which enables rogue or compromised carriers to provision malicious additions to SIMs deployed in the field.

3. Supply-chain attacks. Access to the SIM at any stage between manufacturing and distribution would allow an attacker to implement backdoors on SIM cards.

4. Vulnerabilities in a SIM's software stack. Attackers can exploit vulnerable implementations to compromise the SIM and launch further attacks.

Note that all of these types of attacks were already observed in the wild: SIM interposers are sold to bypass carrier-lock restrictions on iPhones, leaked documents suggest attacks against SIM manufacturers [59], network operators were previously compromised [55], and the recent Simjacker malware leveraged vulnerabilities in a pre-installed SIM Application Toolkit (SAT) cardlets to distribute spyware [60].

Nonetheless, research elaborating on the threats of hostile SIMs and potential defenses is sparse; we partially attribute this to the lack of *security-centric* tooling for SIM card research. While both physical, and minimal software research SIM platforms exist, none of them provide the flexibility (e. g., to holistically violate standards), and tooling required to effectively carry out security research. Nevertheless, previous research in the domain of cellular security highlighted the importance of test beds and versatile tooling: This line of research saw many vulnerabilities discovered only after Software-defined Radio (SDR) projects made radio protocol stacks *modifiable*. With that in mind, this paper introduces tooling that improves accessibility to dynamic testing of SIM-related functionality and baseband implementations in the context of *attacker-controlled* hostile SIMs.[4]

4 Design

We aim to directly support further security research by designing SIMURAI to be *extensible* and *customizable*. While the implementation generally adheres to the relevant standards, our platform does not enforce standards-compliance, as such, it aims to provide researchers with sufficient *flexibility*. Altogether, SIMURAI embodies all core functionality in hopes that it becomes *useful*, *accessible*, and that it contributes towards the development of end-to-end cellular setups.

4.1 Overview

We provide an overview of SIMURAI in Figure 2 and showcases the logical flow of an APDU traversing the framework. At its core, SIMURAI comprises of three main components: *(i)* SWICC, a flexible platform for creating emulated ICCs, *(ii)* SWSIM, which drives SWICC and offers SIM-specific functionality, and *(iii)* an I/O layer which facilitates the connection between a PC/SC-connected SIMURAI, running on a host PC, and UEs.

4.2 SWICC

SWICC is SIMURAI's component for creating various types of smart cards on top of an emulated ICC. It handles all the complexity associated with the transport and application layers, provides a variety of utilities that support processing of APDUs, and uses a Finite State Machine (FSM) to simulate a real ICC device. SWICC also provides a file system which generally follows the standardized definition, while allowing the hierarchy and content definitions to be imported and modified at run-time. It also supports the interindustry class of commands that all ICCs need to recognize, but allows these default implementations to be overridden. Due to the architecture of the APDU demultiplexer (demux), SWICC detects when an instruction should be handled by a proprietary APDU handler. In this case, the message will be delegated, via the APDU demux, to the proprietary handler registered by a user of the framework. Lastly, SWICC offers a *response rewriting* API that allows direct modification of raw R-APDUs with user-defined logic.

[4] i. e., we do not seek out to explore SIM implementations themselves.

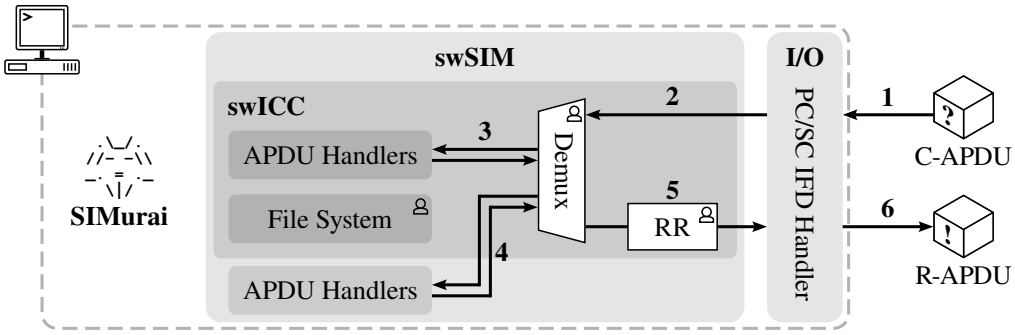

Figure 2: Overview of the SIMURAI architecture, showing the main components and the path of an incoming Command APDU (C-APDU) from the UE to the resulting Response APDU (R-APDU): **1** the PC/SC component provides the interface to, e. g., a physical UE and forwards data to **2** the demultiplexer that routes the command to the appropriate component, i. e. the APDU handlers **3** or **4**, depending on the command. The APDU handlers implement most of the logic. The Response Rewriter (RR) **5** allows users to inspect and override R-APDU contents on the fly. While all code can be modified, the icon denotes components that explicitly allow users to register custom extensions.

4.3 swSIM

SIMURAI leverages swSIM to implement command classes described by ETSI and 3GPP to handle APDUs sent by mobile telecommunication devices such as cellular basebands. swSIM registers the APDU handlers it implements with swICC, effectively turning the ICC into a SIM card by extending its functionality. Furthermore, swSIM provides SIM specific applications, such as the USIM and proactive applications, that can be toggled on and off, or completely overridden with custom implementations of the user.

4.4 I/O Interface

The last part of the SIMURAI platform is the I/O layer that serves as an interface between swICC and the UEs. Its core is the IFD handler, connecting to a PC/SC middleware (e. g., *pcscd*). swICC receives C-APDUs from the PC/SC middleware via a custom network protocol and maps these to one or more Transmission Protocol Data Unit (TPDU) messages. These TPDUs are then sent to the swICC card for processing. Eventually, an R-APDU will be received from swICC and forwarded back to the PC/SC middleware, completing the C-RP. Our architecture allows SIMURAI to flexibly connect to different endpoints. For instance, we can attach swICC-based cards to a PC as if they were physical cards inserted into a smart card reader, or use hardware breakout boards such as the SIMtrace2 to connect to physical COTS UEs.

5 Implementation

The code running on smart cards is considered an implementation detail by the standards describing the ICC, UICC, and SIM. Consequently, it is likely that any two ICC cards will not operate identically, including swICC, any swICC-based smart card, as well as any other SIM card emulator. We implement SIMURAI in 14,449 lines of C code, with swICC accounting for ~55%, swSIM for ~41%, and the PC/SC IFD handler for ~4%.

5.1 swICC

We implement swICC with a variety of utilities and self-contained modules to work with ICC data and behaviors. Additionally, we provide interfaces to ease card development. At the core of swICC sits an FSM tying together all the modules and utilities to parse and handle incoming commands.

Network. The protocol implemented for swICC provides the ability to transfer raw TPDU data, contact states, control requests, and control responses. It is designed such that a receiver obtains a constant-width header, followed by as many data bytes as indicated in the header. Control requests allow for mocking a cold or warm reset of the ICC. This is useful when working with software operating at the application layer, since in this case, contact states and transmission layer details are extraneous. swICC uses its FSM and APDU handlers to determine the length of command data that should be received from the terminal. The responses reuse a one byte control field to indicate success or failure states.

TPDU Parsing. swICC implements a flexible transport layer that communicates an expected data length with the physical layer, but will accept longer and shorter buffers. The number of parts in which a TPDU is received, can be arbitrarily large or small and the number of parts does not influence how the TPDU will be processed. For simplicity, our current

```
{"disk": [{
  "type": "file_mf",
  "id": "3F00",
  "contents":[{
    "type": "file_ef_transparent",
    "id": "2FE2",
    "contents": {
      "type": "hex",
      "contents": "98881201000050018 0F4"
    }}]}]}
```

Listing 1: A simple SWICC file system defined in JSON. It contains one tree (one MF); inside the tree is one file.

implementation of SWICC, offers only the $T = 0$ transmission protocol, drastically reducing complexity as it allows to trivially map between TPDUs and C-APDUs. In regards to our choice of T, to the best of our knowledge, the $T = 1$ transmission protocol is rarely supported by commercial SIMs.

Handling C-APDUs. Within the set of valid command classes, the SWICC implementation only supports the interindustry class, with all others being marked as proprietary (adhering to [27]). The instruction class of incoming commands can contain additional information such as a secure messaging indication, a desired logical channel, or a command chaining indication. SWICC delegates the extraction of additional information from proprietary instruction classes to the proprietary C-APDU handlers registed by the user (e. g., the handlers provided by SWSIM or a user). Each C-APDU handler may decide what procedure bytes to send, and therefore, how to receive command data. Additionally, the C-APDU handler is responsible for tracking sent procedure bytes and ensuring that the correct amount of data is received. SWICC supports these operations by counting procedure bytes, validating universal status words, and buffering data to simplify handling of C-APDUs.

BER-TLV. Many instructions in the universal and interindustry classes encode their responses as BER-TLV objects [12,27]. In order to alleviate the complexity of encoding and decoding such objects, we include an easy-to-use utility as part of SWICC which is directly used by multiple C-APDU handler implementations.

File System. SWICC implements a file system that closely (although not exactly) models the standardized FS for ICCs; we highlight all noteworthy divergences in Section 8. To offer a flexible and human-configurable interface to the FS, SWICC enables the generation of its internal file system from a JSON description. Listing 1 illustrates a small example of a file system definition for SWICC.

5.2 SWSIM

SWSIM implements C-APDU handlers for a minimal set of instructions contained in classes defined by ETSI and 3GPP standards. These classes contain the common instruction set of 3G/4G/5G, and dedicated instructions for 2G. While 2G-only SIMs have a very simple file system, 3G (and newer) SIMs host a more substantial FS hierarchy, with most complexity coming from the introduction of the USIM application (ADF). SIMURAI's SWSIM implementation includes examples of JSON file system definitions for both cases.

Supporting Custom Proactive Commands. SWSIM provides a default proactive application that can be disabled or overridden with a user-defined implementation. When the default application is enabled, the APDU demux of SWSIM will execute a 'step' function provided by the proactive application. This function may generate a new command and place it inside an internal buffer. Subsequently, SWSIM will attempt to override the next R-APDU status word indicating success with the value indicating the availability and length of the proactive command. In case that the terminal does not directly react, SWSIM will repeatedly override status words until the terminal retrieves the proactive command with a FETCH C-APDU.

5.3 I/O: The PC/SC IFD Handler

SIMURAI's IFD handler connects to a PC/SC middleware which exposes an interface to connect—usually physically—card readers as PC/SC devices. The IFD handler takes care of card connections, ensures that C-APDUs get delivered correctly, and that all states presented by the connected card get translated into valid R-APDUs. Commodity PC/SC middleware (e. g., Linux's *pcsc-lite*) requires all drivers to contain a list of supported devices, such that when any of them is attached, the correct driver can be loaded. SIMURAI's virtual reader has no hardware dependencies; therefore, our driver lists `/dev/null` as the only supported device. Because the null device is always present, the PC/SC middleware is tricked into automatically loading the SIMURAI IFD handler.

6 Evaluation

Recall that our goal is to demonstrate not only the advantages of SIMURAI for security research, but also the critical need to consider SIM-based attacks as a potential attack vector for baseband firmware. As such, we set out to answer two main research questions:

RQ1 Can SIMURAI aid security research?

RQ2 Are malicious SIM cards a realistic attack vector?

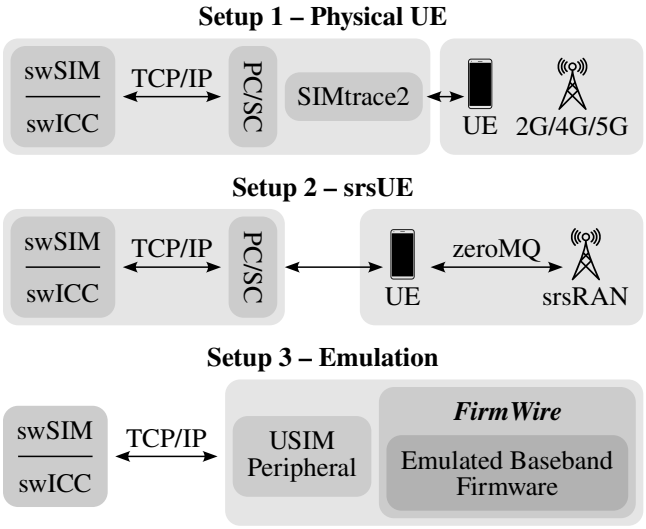

Figure 3: We integrate SIMURAI into three common cellular research setups. Setup 1 consists of 2G (Yate), 4G (srsENB and srsEPC), and 5G (srsGNB and Open5Gs) networks, and COTS UE with SIMtrace2 and cardem-firmware acting as the bridge to PC/SC-attached SIMURAI. Setup 2 contains an emulated srsRAN network and srsUE with SIMURAI attached via PC/SC. Setup 3 integrates SIMURAI with FirmWire, a baseband emulation platform.

To answer *RQ1*, we first integrate SIMURAI in representative cellular research lab setups and connect it to an emulation platform to demonstrate its core functional capabilities (Section 6.1). Then, using these setups, we demonstrate that we can carry out security-relevant experiments. In particular, we first reproduce crucial components of SIM-based spyware and then carry out a fuzzing campaign against commodity baseband implementations (Section 6.2).

To answer *RQ2*, we implement two case studies of attacks originating from a hostile SIM, exemplifying the capabilities of physical attackers and rogue operators (Section 6.3).

6.1 Integration into Cellular Test Beds

We integrate SIMURAI into three common research setups: A physical 2G/4G/5G test bed involving COTS UE, an emulated srsRAN network with srsUE, and FirmWire-emulated basebands. We outline these in Figure 3.

Setup 1: Physical UE in 2G/4G/5G Networks. We perform experiments with multiple COTS smartphones to evaluate if they can successfully communicate with SIMURAI as their SIM. Figure 4 shows our setup which consists of SIMURAI connected directly to commodity smartphones through SIMtrace2. The SIMtrace2 device runs on *cardem* firmware which changes the behavior of the device, from passively tracing communication, to actively intercepting and forwarding messages over USB to a PC/SC-connected card [51]. SIMtrace2 then provides the electrical and transmission-layer interface necessary for valid data exchanges, but does not perform processing of the exchanged APDUs; all SIM-related information is passed to the SIMURAI implementation where we have full control over how it is processed. The networks are based on Yate [45] for 2G, srsENB and srsEPC for 4G [20,62], and srsGNB and Open5Gs [47] for 5G.

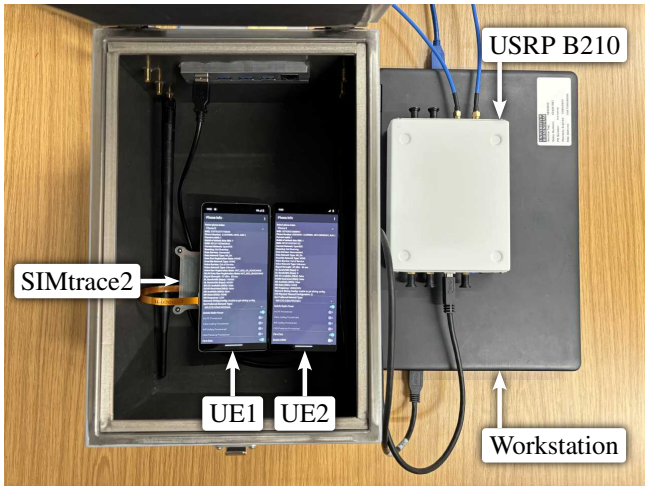

Figure 4: SIMURAI integrated into a 4G/5G cellular test bed.

We verify that each smartphone can successfully connect to our test networks. With respect to SIMURAI, this shows that the smartphone can successfully access the SIMURAI FS when interacting with required files, and that it can use the SIM to run the authentication algorithm (Milenage for 4G/5G, *none* for 2G). Table 2 summarizes our results for all tested smartphones. As demonstrated, SIMURAI proved to be a widely compatible SIM replacement. In practice, this means that SIMURAI contains the files, and supports the subset of instructions required to establish a connection with the network, and provides a functional implementation of Rijndael and Milenage functions. If any of these core components misbehaved, no connection would be established.

Our testing also solidified the fact that no two UE implementations are the same. At first our iPhone 15 tests were unsuccessful because SIMURAI was missing an UPDATE RECORD APDU handler. We added this missing feature and observed no further negative results. All remaining devices only required SIMURAI to be reconfigured with different keys and file contents, no source changes were necessary.

Setup 2: Emulated, SRS-based Network. srsRAN is a comprehensive software framework that provides a nearly complete end-to-end cellular setup. It includes implementations for the core network, an eNodeB, and a UE, making it a

Table 2: List of COTS UEs tested with SIMURAI.

Vendor	Model	Release	SoC	Baseband Version	2G	4G	5G
Apple	iPhone 15	09.2023	A16	1.55.04	●	●	●
	iPhone X	11.2017	A11	8.50.04	●	●	△
Samsung	Galaxy A14	03.2023	S5E3830	A145FXXU4BWK8	●	●	△
	Galaxy S22	02.2022	S5E9925	S901BXXU7DXA6	●	●	●
	Galaxy A52s 5G	09.2021	SM7325	A528BXXS6FXA1	●	●	●
	Galaxy A41	05.2020	MT6768	A415FXXU1ATD1	●	●	△
	Galaxy S10e	03.2019	S5E9820	G970FXXSGHWB3	●	●	△
	Galaxy S9	03.2018	S5E9810	G960FXXS7CSJ3	●	●	△
	Galaxy S7 edge	03.2016	S5E8890	G935FXXU8EUE1	●	●	△
	Galaxy S7	03.2016	MSM8996	G930UUESBCTA3	●	●	△
	Galaxy Core Prime	11.2014	MSM8916	G360FXXU1AOA1	●	●	△
Google	Pixel 7	10.2022	GS201	g5300q-230626-230818-B-10679446	●	●	●
	Pixel 6	10.2021	Tensor	g5123b-116954-230511-B-10112789	●	●	●
OnePlus	Nord CE 2	02.2022	MT6877	M_V3_P10	●	●	●
Oppo	Find X5	02.2022	SM8350	Q_V1_P14	●	●	●
ZTE	Blade A54	09.2020	SC9863A	4G_MODEM_22B_W23.33.3_P3	●	●	△
Oukitel	C19	04.2020	MT6735	E598_37_Q0_LWG_V0.1.2_S200602	●	●	△
Motorola	One Vision	06.2019	S5E9610	S337AP_KANE_SGCS_QB4946070	●	●	△

● : Supported ○ : *Not* supported △ : Network generation not supported by device

preferred choice for research setups. Similarly to *Setup 1*, the UE component, srsUE, can integrate with a card reader to access hardware SIM. This capability allows for the integration of SIMURAI via SIMtrace2. To further test the versatility of SIMURAI, we attach it directly to the SIM layer of the UE implementation, srsUE, using its PC/SC interface. This direct attachment bypasses the need for any hardware, facilitating the direct exchange of APDUs between the UE and SIM. This integration was straightforward and successful. The addition of SIMURAI represents a step towards achieving a fully virtual, end-to-end cellular setup.

Setup 3: Emulation Platform. Our third experimental setup integrates SIMURAI into FirmWire [22], a recently published full-system emulation platform for baseband firmware images. The authors of [22] explicitly note that *"to enable a fully virtual UE, USIM peripheral support would need to be prototyped into FirmWire. This would yield more accurate processing of messages, especially those which require a SIM card such as SMS or USSD"*. Hence, we believe that integrating SIMURAI with this emulation platform can play a crucial part in fully virtualizing COTS UEs for security testing.

Unfortunately, unlike the tools discussed in the previous two case studies, FirmWire lacks a PC/SC interface or any means of connection a physical or virtual SIM, further compounding the issue. Therefore, we reverse engineered the baseband firmware for Samsung Exynos-based UEs supported by FirmWire and identified how it would interact with a physical SIM card. Based on our insights, we implemented a USIM peripheral, which uses SIMURAI's low-level interfaces to exchange TPDUs between our platform and FirmWire.

Our USIM peripheral implementation, consisting of 299 lines of Python code, enables successful data exchange between the frameworks at the TPDU level. We experimentally verified compatibility of our SIMURAI integration for the different Shannon-based UEs supported by FirmWire and present the results in Table 3. The log messages of FirmWire indicate a successful initialization of the baseband's USIM task and elaborate accesses to SIMURAI's file system for all successfully tested firmware versions. In the case of firmware targeting Samsung Galaxy S7 and Samsung Galaxy S7 edge phones, the interaction between the firmware and USIM peripheral changed sufficiently to render our current peripheral implementation inoperable. While we expected that a single USIM peripheral is unlikely to work with all emulated UEs due to changes in hardware and firmware, we note that our implementation already spans multiple phone models and generations, indicating that physical USIM peripherals are rarely subject to drastic changes.

6.2 SIMURAI as a Research Platform

6.2.1 Simulating SIM Spyware

Previous research has shown the potential for spyware-like features to be embedded in SIM applications. Notably, Monkeycalendar and Gopherset are two examples of SIM-based

Table 3: Baseband firmwares tested for SIMURAI support through a USIM peripheral.

Model	Chipset	FW Version	Result
One Vision	S337AP	RSA31.Q1-48-36-23	●
S10	G973F	CP_G973FXXU9FUCD	●
S10e	G970F	CP_G970FXXSGHWC2	●
S9	G960F	CP_G960FXXUGFUG4	●
S8	G950	CP_G950FXXU1AQI7	●
S8+	G955	CP_G955FXXU1AQF7	●
S7Edge	G935	CP_G935FXXU8ETI2	○
S7	G930	CP_G930FXXU8ETI2	○

spyware identified in the NSA's ANT catalog (which became public in 2013) [6]. These SIM-based spywares exfiltrate sensitive information—such as a victim's location, phonebook, SMS messages, and call logs—using SMS SUBMIT messages. A similar technique was observed in the more recent Simjacker attack [41, 60]. We chose to re-implement the crucial part of the Simjacker spyware to test the adaptability of SIMURAI. Our intuition is that an open-source mimicry implementation of the spyware can help the research community to assess the severity of this threat and investigate tractable countermeasures, especially as SIMURAI can closely monitor and modify individual operations.

For our implementation, we use *Setup 1* described in Section 6.1. The core functionality of the spyware, i. e., location retrieval and data exfiltration via SMS, is realized using proactive commands (i. e., PROVIDE LOCAL INFORMATION and SEND SHORT MESSAGE). We extended SIMURAI to send these proactive commands to the UE, as shown by Figure 5, and could observe the location information being sent via the network to the second UE. The ease of re-implementing this spyware—overall, our proof-of-concept consists only of 89 lines of C code—demonstrates not only are there severe privacy issues when considering malicious SIM cards as an attack vector, but also the upside of having a research platform to evaluate attacks in a quick and flexible manner.

6.2.2 Fuzzing UE Implementations

In the last decade, fuzzing has become one of the most prevalent techniques for uncovering implementation flaws, a trend which is also noticeable in cellular security research [17, 18, 22, 40, 52]. Hence, we set out to use SIMURAI to enable an efficient fuzzing setup to test the robustness of proactive command handling embedded in commodity baseband firmware. We deliberately chose these proactive commands as the fuzz target because: They can be triggered directly from a malicious SIM card, they provide a variety of different functionality, and they require complex parsers for many different binary formats, as outlined in Section 2.2.

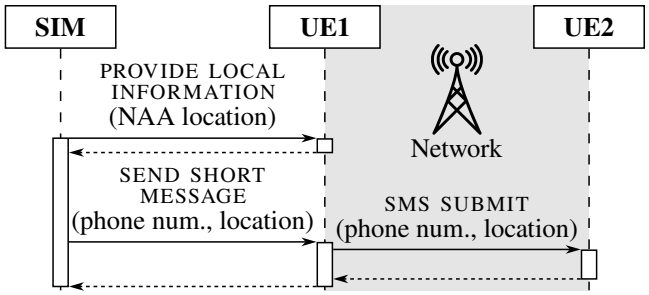

Figure 5: Message exchange for the location-stealing spyware. The SIM implementation is provided by SIMURAI.

Attempt 1: Physical Setup. Our first, naïve approach was to connect a physical phone to SIMURAI (i. e., using *Setup 1*). Unfortunately, our experiments on a Motorola One Vision[5] quickly unveiled that terminals throttle the connection to the SIM when repeatedly receiving malformed data, resulting in execution speeds of less than one test case per second. Additionally, the lack of fast reset mechanisms and transparent feedback channels created further challenges: We cannot know the state of the target when a given input is processed, and without additional feedback, such as coverage, we cannot rank inputs for further mutation and testing.

We conclude that while technically possible, fuzzing in this case proved limited and ineffective.

Attempt 2: Emulated Setup. Given the roadblocks of physical fuzzing, we decided to leverage emulation, i. e., our *Setup 3* consisting of SIMURAI connected to FirmWire. With this setup, we tested the most recent firmware available for the Motorola One Vision[6] in a long-lasting fuzzing campaign. We used AFL++ v4.09a [16] as the fuzzer and distributed the fuzzing efforts across 10 cores on an AMD EPYC 7662 64-Core CPU running Ubuntu 18.04; we fuzzed the target for 18 days. Additionally, we used SIMURAI to generate an expansive multi-thousand entry corpus of valid proactive commands to be used as seed inputs.

Approach. Implementing a fuzzing campaign against the proactive command handlers in FirmWire is not straightforward. The fuzzing methodology of the emulation platform expects that fuzzers are implemented as *tasks* for the emulated baseband real-time operating system, and that fuzzing inputs are sent via the operating system's message queues. However, we noticed that sending messages directly to the *USAT* task, which dispatches incoming proactive commands, would fail because the task was missing important state initialization.

As a result, we decided to pivot our fuzzing to a hybrid approach: We use SIMURAI to serve SIM-related requests to

[5] M3BA0, Baseband S337AP_KANE_SGCS_QB4946070
[6] XT1970-3_KANE_RETEU_11_RSA31.Q1-48-36-23

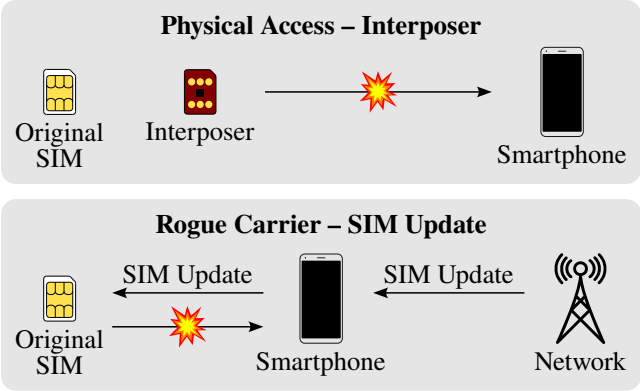

Figure 6: We use two separate setups to demonstrate the feasibility of SIM-originating attacks against basebands. The Interposer in the first setup is inserted between original SIM and smartphone. In the second setup, we update a SIM card over-the-air to showcase the threat of *rogue carriers*.

FirmWire until initialization is completed, and the emulated baseband would assume the presence of a SIM card. Then, we create a snapshot of the emulated baseband state and use it as basis for fuzzing. During fuzzing, we prefix the fuzzing input with the baseband internal indicator for proactive commands and provide this modified input to the USAT task. We note that this approach would also be possible by connecting a physical SIM, via SIMtrace2, to FirmWire for initialization. However, before our work, FirmWire neither exposed a PC/SC interface nor included a USIM peripheral, preventing the use of SIMtrace2 altogether. Additionally, eliminating the need for hardware eases the creation of powerful and parallelizable fuzzing setups.

Results. During our fuzzing campaign, the fuzzing instances cumulatively executed 490 million test cases and found 63 crashes. After manual deduplication, we could narrow down the root cause for the crashes to two distinct vulnerabilities: *(i)* A null-pointer dereference during handling of the SEND SMS command, and *(ii)* a heap buffer overflow during handling of the SEND SS proactive command.

To verify whether the discovered vulnerabilities were present on more recent devices, we replayed the crashing inputs using SIMURAI on Google Pixel 6 and Google Pixel 8 devices. We could verify the presence of vulnerability *(i)* on both devices, and vulnerability *(ii)* on the Pixel 6. Interestingly, for vulnerability *(ii)*, the device would, with some inputs, also crash indicating a PREFETCH ABORT error message, which indicates a possible corruption of the program counter. Both vulnerabilities were acknowledged by Google as high-severity, and got assigned the following identifiers: CVE-2023-50806 and CVE-2024-27209.

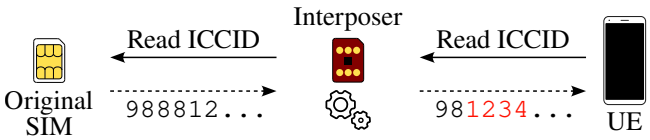

Figure 7: An interposer intercepts and selectively manipulates the communication between a phone and SIM card.

6.3 SIM Attack Case Studies

In the previous section, we find vulnerabilities in baseband implementation reachable through malicious, SIM-originating, messages. Consequently, we want to assess and demonstrate the practical feasibility of these attacks. Figure 6 summarizes our case studies.

6.3.1 Physical Access: Interposer

Threat Model. For our first attack case study, we assume an attacker with short-term physical access to the victim's phone. While this may appear like a strong attacker, this scenario can be relatively easily achieved for highly targeted attacks.

The Attack. We base our attack on a SIM interposer; a "thin SIM" that is positioned between the SIM card and the phone's card reader, as shown in Figure 7. The attacker could insert it by gaining access to the phone's SIM slot. We found that certain commercially available interposers are *(i)* updateable using a smart card reader, *(ii)* the update files contain the raw, unencrypted, and unobfuscated, firmware of the interposer, and *(iii)* the card does not perform any integrity checks on the received firmware update.

We found out that the interposer runs an unidentified RISC-V CPU, and through analysis of the update process, we found commands that: Boot the chip into a bootloader, delete memory regions, and flash new firmware. We were able to analyze the firmware, particularly the portion that performs the interception of SIM communication. With that knowledge, we create a patched firmware that triggers the vulnerability we found in Section 6.2.2. Overall, our binary patches required the modification of 57 bytes to trigger vulnerability *(i)*, and 42 bytes to trigger vulnerability *(ii)*, with the core difference being the crashing proactive command itself.

Evaluation. We prepare the patched firmware and flash it to an interposer. We open the SIM tray, sandwich the interposer and SIM together, and close the SIM tray with both the original SIM and the interposer attached to the phone's SIM reader. In our experience, this takes only a few seconds and requires no further manipulation of the SIM tray (i. e., it fits well inside all devices we were able to test).

This experiment showcases that anybody with physical access can sneak-in the interposer inside the phone, and potentially launch attacks from this platform. In our proof-of-concept interposer firmware, the vulnerability is triggered immediately when the phone accesses the SIM card; in some cases (e. g., when the phone is powered off), it might happen with a delay. A more complex firmware patch could allow a deliberate, attacker-controlled delay. Usually, interposers only inject or manipulate selectively, and pass other SIM card communication through. Thereby, the attack could be launched without impeding the functionality of the original SIM card.

6.3.2 Rogue Carrier: Over-the-Air SIM Card Update

Threat Model. For our second case study, we showcase the capabilities of rogue carriers in more detail. From a user's perspective, there are multiple possible ways to encounter such carriers, for instance via insider threads, supply chain attacks, or law enforcement with access to the network.

The Attack. Carriers can update both the file system content and applications on a SIM card *remotely* [1]. These procedures are secured with pre-shared keys, programmed into the UICC and known to the operator. To remotely load applets onto a SIM card, the operator can use a special class of Short Messages (SMS), commonly known as *binary SMS*. While this update procedure is well-known and documented by the academic community [35], we could not find any openly available implementations of remote OTA updates.

Remote SIM Updates. The SMS header indicates that the message is addressed to the SIM card (rather than the phone, like a regular text message). The phone forwards the contents of the SMS to the SIM and thereby acts like a remote card reader. It is important to note that the SMS payload is encrypted and integrity protected, so that only the issuer of the SIM card can run remote APDU commands and thereby push updates to the SIM. Generally, the remote update procedure is similar to an update through a card reader. Figure 8 shows the encapsulation of APDUs within an SMS-DELIVER packet. The phone wraps the whole SMS into a GlobalPlatform ENVELOPE command and sends it to the SIM card. Locally, one would directly send the ENVELOPE with the SMS to the SIM.

Setup. Our setup consists of an srsEPC core network and an srseNB base station connecting to a LimeSDR Mini SDR inside a faraday cage. We modify srsEPC to enable SMS delivery through NAS Downlink Messages. The core network is greatly simplified in our experiment, i. e., commercial networks would typically use a separate entity that manages the delivery of SMS. Furthermore, we use the most modern and simple form of SMS delivery using NAS Downlink Messages. Other ways exist (e. g., SMS-over-IMS); however, these differences do not affect the OTA update mechanism.

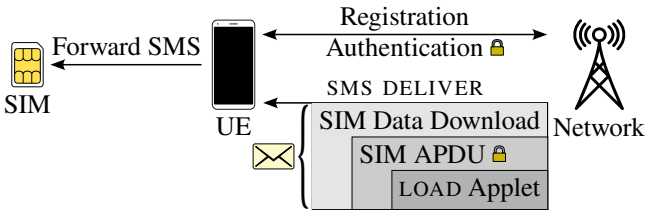

Figure 8: Process of installing a cardlet OTA.

Evaluation. We prepare a SIM applet that triggers the vulnerability through a proactive command, and compile that applet into the Java Card CAP format. We then identify the targeted card to retrieve the necessary keys to update the SIM. Using the keys, we create encrypted, SMS-wrapped APDUs that load the applet chunk by chunk onto the card. These APDUs are then fed to the core network, which encapsulates the SMS and delivers them to the SIM card.

In our experiment, the remote installation was successful, and the applet successfully sent the malicious commands to the baseband. The experiment demonstrates that a rogue carrier—as the issuer of the SIM card inside the phone—can push remote updates and trigger the SIM-exposed baseband vulnerability. After successful installation, we can remotely remove the applet from the SIM again to make the entire operation more covert.

7 Mitigation

We are not aware of mitigations currently in place to inhibit the threat of SIM-originating attacks under our assumed threat model of rogue carriers or with physical access. The inclusion of SIM-originating attacks into threat models would justify the hardening of cellular basebands, as called for by Google [38, 39], and thereby reduce the susceptibility of this attack surface in the future.

Currently proposed, SIM-related security advancements largely target aspects other than baseband security: Jinghao et al. propose to secure the channel between SIM card and phone to stop interposers from intercepting and manipulating the data in transit [66]. If deployed and enforced, this would indeed render the local attacker with physical access ineffective. The SecureSIM scheme, however, is primarily designed to protect the SIM card, not the baseband. Hence, it would require further evaluation if the scheme could also protect the baseband or if the interposer could launch an attack *before* the (secured) communication with the original SIM starts. The introduction of the *eSIM* is sometimes associated with gains in security—and indeed, an *embedded SIM* does not *easily* allow placing an interposer inbetween (however, many eUICC are *surface-mounted*, so access is not impossible). While this certainly raises the bar for physical attackers, the issue with

potential rogue carriers remains, as the eSIM has the same baseband communication capabilities and is also controlled by the carrier. RILDefender, Simjacker, and others discuss threats related to binary SMS—and call for filtering by the carrier or in the Android system [60,65]. While these defenses work against *third-party*-originating SMS, they do not protect against the *rogue carrier* that we assume.

One effective mitigation could be to *disable the processing of proactive commands* in the baseband. This mostly reduces the functionality of SIM cards to a file system and breaks legitimate use-cases of proactive commands, but greatly reduces the available attack surface. If basebands expose that option to the smartphone operating system, the smartphone could selectively disable them. Similarly to the option to disable 2G, disabling proactive commands could be a user-facing option, or part of a *lockdown mode* as found in iOS.

Additionally, during our analysis of the vulnerabilities, we noticed that commodity baseband firmware appears to ship code that—according to the TERMINAL PROFILE—is not supported, and, hence, *never* needed by the smartphone. Therefore, we think that even without reducing functionality, basebands have potential for de-bloating the code shipped within the firmware. Similarly to Google's call for updating the build toolchains [38], baseband firmware developers and integrators could disable unused features at build time.

8 Discussion

Scalability. Security tooling, especially when used for fuzzing, needs to allow for scalability and SIMURAI is no exception. In physical setups, the main bottleneck for scalability is the required hardware: Each instance of SIMURAI would require a dedicated SIMtrace2 device to connect to a UE. For emulated settings, however, each emulator instance can be paired to a unique instance of SIMURAI, allowing for easy scaling. The main roadblock to scaling emulated fuzzing to arbitrary UEs is FirmWire support and the need to create a USIM peripheral for the targeted hardware platform. This requires a significant amount of manual engineering and reverse engineering efforts. Nevertheless, as shown in Table 3, a single implementation is likely to cover multiple generations of UEs from a given vendor.

Realism of Presented Attacks. In our experiments, we show two example attacker models leveraging hostile SIMs. Attackers with physical access can use an interposer to inject an arbitrary payload into the communication between the phone and the original SIM card. Rogue carriers can use their ability to update SIM cards remotely to install malicious SIM card applets and launch the attack from there.

We cannot assess the real-world possibility of obtaining physical access to relevant phones or access to remote SIM management. However, previous incidents demonstrated misuse of telecom infrastructure for attacks against subscribers [55, 59] and the *technical* challenge of programming an interposer for physical access is relatively low. In our experiments, SIMURAI allowed for quick prototyping of proof-of-concept attack implementations for both scenarios, showcasing the threats posed by hostile SIMs.

Additional Proactive Attack Surface. During our fuzzing and dissection of proactive commands, we found two additional curiosities with security relevance. First, the standard explicitly defines an OPEN BROWSER proactive command, which would attempt to open a browser visiting a specified URL. We found this proactive command to be supported by most modern COTS UEs, with the caveat that the user would be asked for confirmation to open the browser. However, the text in the notification asking for confirmation is completely SIM-controlled. Given that modern smartphone exploit chains often require victims to visit attacker-controlled websites in the process, this feature could provide an additional 1-click exploit attack surface.

Second, the standard also defines a RUN AT proactive command which would directly execute a SIM-specified AT command on the UE. Previous research has shown that attacker-controlled AT commands can have severe consequences (e. g., lock screen bypass [64] or flashing of compromised firmware [53]). Although none of the UEs we tested reported support for this proactive command in their terminal profiles, we still found the code paths handling this command in the baseband firmware images we reverse engineered. Even if this code is not reached under normal operation, we argue that its presence introduces an unnecessary attack surface which could be used by attackers during exploitation. Given that the UEs do not report support for this feature, we believe that debloating according parts of the baseband firmware to remove unsupported parts would be a viable step to improve its overall security, as discussed in Section 7.

Limitations. The SWICC file system does not model the ICC file system exactly. SWICC considers File IDentifiers (FIDs) globally unique. Unfortunately, we later observed that due to FID reuse between DF.GSM and ADF.USIM, this prevented SIMURAI from supporting both DF.GSM and ADF.USIM at the same time. Another limitation is that SIMURAI is not designed to be a replacement for production SIMs, as it lacks security features like file access conditions.

The core design of SIMURAI also relies on a maximum message length aligned with regular APDUs, this makes it difficult to extend the design to support extended APDUs, i. e., APDU messages with a maximum length of around 65k bytes. For a similar reason, we believe that implementing additional transmission protocols would also be challenging.

9 Related Work

Cellular Security. Multiple studies focus on cellular standards, either by studying the standard itself or the implementation of its procedures. Some authors model the cellular standard and compare that with models inferred from an implementation, showing implementation issues and deviations from security goals [9, 23, 24, 29, 32, 34, 52]. Hussain et al. identify that many previously discovered flaws are based on an insecure connection establishment procedure [56]. In an overview study, Rupprecht et al. analyze the various protocol weaknesses and identify several root causes [58].

Software SIM. A notable development in simulation of smart cards is the VSmartCard project [43], which provides an implementation of a generic ICC. In that regard, it is similar to our swICC component. However, we consider swICC to be an improvement as it supports file system persistence and TPDU-level communication, while VSmartCard operates only at the application layer with APDUs. Concurrent to our work, the Onomondo SoftSIM, a software-only SIM implementation, was released [46]. SoftSIM provides a partially overlapping feature set with SWSIM, but primarily targets the needs of commercial IoT vendors rather than security research. vSIM by Kasper & Kuntze et al. [30] is a theoretical software SIM architecture which relies on a trusted platform module (TPM) to achieve security guarantees similar to those offered by a physical SIM card. The authors conclude that a software-only SIM is possible, but they do not provide a publicly available prototype. SIMulator [57] is one notable SIM simulation project that uses a hybrid approach where a real SIM card would process most commands, and only selected commands would be intercepted and handled in software. While this does indeed fulfill the need to selectively manipulate commands, requiring a hardware SIM limits scalability, especially with the fuzzing use case in mind.

SIM-Related Security. To our knowledge, there are only a few academic works in the context of SIM cards. Most related to our work is a study by Jinghao et al. that identifies the issue of unprotected SIM-smartphone communication, and designs an improved access control for SIM cards [66]. While they do identify interposers as potential threats, they consider them a threat to the secrets stored on the SIM card. In contrast, we use them to highlight the risks *towards the baseband*. Related to the capabilities that smartphones expose via binary SMS, Wen et al. propose RILDefender to detect and block SMS attacks in Android's Radio Interface Layer (RIL) [65]. In our case, the harmful payload could not be filtered at the RIL as it comes directly from the SIM.

There are notable contributions at non-academic venues that discuss SIM-related threats. The "Simjacker" analysis exposes multiple on-SIM applets as vulnerable to remote code execution, and shows how this allowed localization of phones [60]. ShadySIM [35, 49] by Koscher and Butler provides a valuable tool for provisioning of applets. We use Simjacker as an example and implement a similar spyware within our SIMURAI framework to evaluate its extensibility. The same author presents an overview of past attacks that used *binary SMS* and calls, for better protection of SIM applets, and network-side filtering [41]. Nohl points out that OTA updates may use weak protection and, among other measures, calls for carriers to filter binary SMS from unknown sources [44]. Burgess reports the use of proactive commands on various carrier applets [7]. In a talk at 35C3, Sesterhenn finds Linux driver vulnerabilities triggered by a smart card [61].

Baseband Fuzzing. Fuzzing work in context of baseband firmware include ARIstoteles [36], LTEFuzz [33], Berserker [54] and 5Ghoul [18]. Besides this, re-hosting [15] is a relatively new approach for security analysis of embedded systems, which enabled fuzzing of emulated baseband firmware as shown with Basesafe [40] and FirmWire [22]. In contrast to SIMURAI, none of these approaches consider the SIM card as an attack vector in their fuzzing setup.

Mobile Ecosystem. Many aspects of cellular network security are defined by non-technical phenomena: Lee et al. find that flaws within customer authentication processes of US-based carriers enable *SIM swapping*, i. e., associating a phone number with a new SIM with the goal of identity fraud [37]. McDonald et al. analyze the issues users face with phone numbers being a de-facto identifier for online services [42].

10 Conclusion

In this paper, we argued for the inclusion of hostile SIMs, as an attack vector, in cellular security threat models. Various scenarios can lead to a compromised SIM, including rogue operators, physical attackers, and vulnerabilities in the SIM itself. To demonstrate the importance of this revised threat model and to facilitate additional research, we design and implement SIMURAI, a first-of-its-kind research platform that provides security analysts with flexible control over SIM card communication.

We integrate SIMURAI in common cellular test bed setups and demonstrated its capabilities for security research: Among others, we showed how SIMURAI enabled us to perform a fuzzing campaign against COTS UE implementations leading to the discovery of two high-severity security vulnerabilities in recent Google smartphones, affecting millions of devices.

Finally, we demonstrated how various attackers could exploit hostile SIMs to trigger the identified vulnerabilities, and we proposed potential mitigations to address these issues.

Coordinated Disclosure

We reported the vulnerabilities identified during our research to Google. These vulnerabilities were assigned CVE-2023-50806 and CVE-2024-27209 with patches released as part of the Google Pixel updates in March 2024.

Acknowledgements

We would like to thank the anonymous reviewers and shepherd for their insightful and constructive comments. We would further like to thank Bedran Karakoc, Daniel Klischies, and Dominik Maier (in no particular order) for their help, comments, and discussion. This research was funded in part by UKRI EP/V000454/1 and supported via an Android Security and PrIvacy REsearch (ASPIRE) Award from Google. The results feed into DSbDtech.

References

[1] 3GPP. Security mechanisms for the (U)SIM application toolkit; Stage 2. TS 23.048 V5.9.0.

[2] 3GPP. Subscriber Identity Module Application Programming Interface (SIM API) for Java Card™. TS 03.19 V8.5.0.

[3] 3GPP. Universal Subscriber Identity Module (USIM) Application Toolkit (USAT). TS 31.111 V18.5.1.

[4] 3GPP. (U)SIM Application Programming Interface (API); (U)SIM API for Java™ Card. TS 31.130 V18.0.0.

[5] Zaheer Ahmad, Lishoy Francis, Tansir Ahmed, Christopher Lobodzinski, Dev Audsin, and Peng Jiang. Enhancing the Security of Mobile Applications by Using TEE and (U)SIM. In *IEEE Conference on Ubiquitous Intelligence and Computing and Conference on Autonomic and Trusted Computing (UIC-ATC)*, 2013.

[6] Jacob Applebaum, Judith Horchert, Ole Reissmann, Marcel Rosenbach, Jörg Schindler, and Christian Stöcker. NSA's Secret Toolbox: Unit Offers Spy Gadgets for Every Need. Spiegel International, spiegel.de, 2013.

[7] David Burgess. Proactive SIMs. DeepSec 2021, deepsec.net, 2021.

[8] Dhiman Chakraborty, Lucjan Hanzlik, and Sven Bugiel. simTPM: User-centric TPM for Mobile Devices. In *USENIX Security Symposium*, 2019.

[9] Yi Chen, Yepeng Yao, XiaoFeng Wang, Dandan Xu, Chang Yue, Xiaozhong Liu, Kai Chen, Haixu Tang, and Baoxu Liu. Bookworm Game: Automatic Discovery of LTE Vulnerabilities Through Documentation Analysis. In *IEEE Symposium on Security and Privacy (S&P)*, 2021.

[10] ETSI. Smart Cards; Card Application Toolkit (CAT). TS 102 223 V17.3.0.

[11] ETSI. Smart Cards; Embedded UICC; Requirements Specification. TS 103 383 V14.0.0.

[12] ETSI. Smart Cards; UICC-Terminal interface; Physical and logical characteristics. TS 102 221 V18.1.0.

[13] ETSI. Smart Cards; Vocabulary for Smart Card Platform specifications. TR 102 216 V5.1.0.

[14] ETSI. (U)SIM Application Programming Interface (API); (U)SIM API for Java™ Card. TS 131 130 V18.0.0.

[15] Andrew Fasano, Tiemoko Ballo, Marius Muench, Tim Leek, Alexander Bulekov, Brendan Dolan-Gavitt, Manuel Egele, Aurélien Francillon, Long Lu, Nick Gregory, Davide Balzarotti, and William Robertson. SoK: Enabling Security Analyses of Embedded Systems via Rehosting. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2021.

[16] Andrea Fioraldi, Dominik Maier, Heiko Eißfeldt, and Marc Heuse. AFL++: Combining Incremental Steps of Fuzzing Research. In *USENIX Workshop on Offensive Technologies (WOOT)*, 2020.

[17] Matheus E Garbelini, Zewen Shang, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. Towards Automated Fuzzing of 4G/5G Protocol Implementations Over the Air. In *IEEE Global Communications Conference (GLOBECOM)*, 2022.

[18] Matheus E. Garbelini, Zewen Shang, Shijie Luo, Sudipta Chattopadhyay, Sumei, and Ernest Kurniawan. 5GHOUL: Unleashing Chaos on 5G Edge Devices. Technical report, Singapore University of Technology and Design (SUTD) and I2R, A*STAR, 2023.

[19] GlobalPlatform. Card Specification V2.3.1.

[20] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D Sutton, Pablo Serrano, Cristina Cano, and Doug J Leith. srsLTE: an open-source platform for LTE evolution and experimentation. In *ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH)*, 2016.

[21] GSMA. Common Implementation Guide to Using the SIM as a 'Root of Trust' to Secure IoT Applications. gsma.com, 2019.

[22] Grant Hernandez, Marius Muench, Dominik Christian Maier, Alyssa Milburn, Shinjo Park, Tobias Scharnowski, Tyler Tucker, Patrick Traynor, and Kevin R. B. Butler. FirmWire: Transparent Dynamic Analysis for Cellular Baseband Firmware. In *Symposium on Network and Distributed System Security (NDSS)*, 2022.

[23] Syed Rafiul Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. LTEInspector: A Systematic Approach for Adversarial Testing of 4G LTE. In *Symposium on Network and Distributed System Security (NDSS)*, 2018.

[24] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 5GReasoner: A Property-Directed Security and Privacy Analysis Framework for 5G Cellular Network Protocol. In *ACM Conference on Computer and Communications Security (CCS)*, 2019.

[25] Syed Rafiul Hussain, Imtiaz Karim, Abdullah Al Ishtiaq, Omar Chowdhury, and Elisa Bertino. Noncompliance as Deviant Behavior: An Automated Black-box Noncompliance Checker for 4G LTE Cellular Devices. In *ACM Conference on Computer and Communications Security (CCS)*, 2021.

[26] ISO. Identification cards – Integrated circuit cards – Part 3: Cards with contacts – Electrical interface and transmission protocols. ISO/IEC 7816-3:2006(E).

[27] ISO. Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange. ISO/IEC 7816-4:2020(E).

[28] ITU. Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). ITU-T X.690.

[29] Imtiaz Karim, Syed Rafiul Hussain, and Elisa Bertino. ProChecker: An Automated Security and Privacy Analysis Framework for 4G LTE Protocol Implementations. In *IEEE Conference on Distributed Computing Systems (DCS)*, 2021.

[30] Michael Kasper, Nicolai Kuntze, and Andreas U. Schmidt. Subscriber Authentication in Cellular Networks with Trusted Virtual SIMs. In *IEEE Conference on Advanced Communication Technology (ICACT)*, 2008.

[31] Eunsoo Kim, Min Woo Baek, CheolJun Park, Dongkwan Kim, Yongdae Kim, and Insu Yun. BASECOMP: A Comparative Analysis for Integrity Protection in Cellular Baseband Software. In *USENIX Security Symposium*, 2023.

[32] Eunsoo Kim, Dongkwan Kim, CheolJun Park, Insu Yun, and Yongdae Kim. BASESPEC: Comparative Analysis of Baseband Software and Cellular Specifications for L3 Protocols. In *Symposium on Network and Distributed System Security (NDSS)*, 2021.

[33] Hongil Kim, Jiho Lee, Eunkyu Lee, and Yongdae Kim. Touching the Untouchables: Dynamic Security Analysis of the LTE Control Plane. In *IEEE Symposium on Security and Privacy (S&P)*, 2019.

[34] Daniel Klischies, Moritz Schloegel, Tobias Scharnowski, Mikhail Bogodukhov, David Rupprecht, and Veelasha Moonsamy. Instructions Unclear: Undefined Behaviour in Cellular Network Specifications. In *USENIX Security Symposium*, 2023.

[35] Karl Koscher and Eric Butler. The Secret Life of SIM Cards. DEF CON, 2013.

[36] Tobias Kröll, Stephan Kleber, Frank Kargl, Matthias Hollick, and Jiska Classen. ARIstoteles – Dissecting Apple's Baseband Interface. In *European Symposium on Research in Computer Security (ESORICS)*, 2021.

[37] Kevin Lee, Benjamin Kaiser, Jonathan Mayer, and Arvind Narayanan. An Empirical Study of Wireless Carrier Authentication for SIM Swaps. In *USENIX Symposium on Usable Privacy and Security (SOUPS)*, 2020.

[38] Ivan Lozano and Roger Piqueras Jover. Hardening cellular basebands in Android. Google Security Blog, google-blog.com, 2023.

[39] Ivan Lozano, Roger Piqueras Jover, Sudhi Herle, and Stephan Somogyi. Hardening Firmware Across the Android Ecosystem. Google Security Blog, google-blog.com, 2023.

[40] Dominik Maier, Lukas Seidel, and Shinjo Park. BaseSAFE: Baseband SAnitized Fuzzing through Emulation. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WISEC)*, 2020.

[41] Cathal Mc Daid. STK, A-OK? Mobile Messaging Attacks On Vulnerable SIMs. Virus Bulletin Conference, vblocalhost.com, 2021.

[42] Allison McDonald, Carlo Sugatan, Tamy Guberek, and Florian Schaub. The Annoying, the Disturbing, and the Weird: Challenges with Phone Numbers as Identifiers and Phone Number Recycling. In *ACM Conference on Human Factors in Computing Systems (CHI)*, 2021.

[43] Frank Morgner and Dominik Oepen. VSmartCard. github.com. [Online, accessed June 19, 2024].

[44] Karsten Nohl. Rooting SIM cards. Blackhat 2013, blackhat.com, 2013.

[45] Null Team Impex. YateBTS. yatebts.com. [Online, accessed June 19, 2024].

[46] onomondo. Onomondo UICC. github.com. [Online, accessed June 19, 2024].

[47] Open5GS. Open Source Project of 5GC and EPC. open5gs.org. [Online, accessed June 19, 2024].

[48] Osmocom Contributors. pySIM. Osmocom Wiki, osmocom.org. [Online, accessed June 19, 2024].

[49] Osmocom Contributors. shadySIM. Osmocom Wiki, osmocom.org. [Online, accessed June 19, 2024].

[50] Osmocom Contributors. SIMtrace2. Osmocom Wiki, osmocom.org. [Online, accessed June 19, 2024].

[51] Osmocom Contributors. SIMtrace2 Cardem. Osmocom Wiki, osmocom.org. [Online, accessed June 19, 2024].

[52] CheolJun Park, Sangwook Bae, BeomSeok Oh, Jiho Lee, Eunkyu Lee, Insu Yun, and Yongdae Kim. DoLTEst: In-depth Downlink Negative Testing Framework for LTE Devices. In *USENIX Security Symposium*, 2022.

[53] André Pereira, Manuel Correia, and Pedro Brandão. USB Connection Vulnerabilities on Android Smartphones: Default and Vendors' Customizations. In *IFIP Communications and Multimedia Security (CMS)*, 2014.

[54] Srinath Potnuru and Prajwol Kumar Nakarmi. Berserker: ASN.1-based Fuzzing of Radio Resource Control Protocol for 4G and 5G. In *IEEE Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2021.

[55] Vassilis Prevelakis and Diomidis Spinellis. The Athens Affair. IEEE Spectrum, spectrum.ieee.org, 2007.

[56] Syed Rafiul Hussain, Mitziu Echeverria, Ankush Singla, Omar Chowdhury, and Elisa Bertino. Insecure Connection Bootstrapping in Cellular Networks: The Root of All Evil. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WISEC)*, 2019.

[57] Sebastian Renner and Enrico Pozzobon. SIM Simulator. Troopers'19, troopers.de, 2019.

[58] David Rupprecht, Adrian Dabrowski, Thorsten Holz, Edgar Weippl, and Christina Pöpper. On Security Research towards Future Mobile Network Generations. *IEEE Communications Surveys & Tutorials*, 20(3):2518–2542, 2018.

[59] Jeremy Scahill and Josh Begley. The Great SIM Heist: How Spies Stole the Keys to the Encryption Castle. The Intercept, theintercept.com, 2015.

[60] Adaptive Mobile Security. Simjacker Technical Paper. enea.com, Version 10OCT19-v1.01, 2019.

[61] Eric Sesterhenn. In Soviet Russia Smart Card Hacks You. Chaos Communication Congress 35, media.ccc.de, 2018.

[62] srsRAN Project. Open Source RAN. srsran.com. [Online, accessed June 19, 2024].

[63] sysmocom. sysmoISIM-SJA2 programmable SIM/USIM/ISIM cards. sysmocom.de. [Online, accessed June 19, 2024].

[64] Dave (Jing) Tian, Grant Hernandez, Joseph I. Choi, Vanessa Frost, Christie Raules, Patrick Traynor, Hayawardh Vijayakumar, Lee Harrison, Amir Rahmati, Michael Grace, and Kevin R. B. Butler. ATtention Spanned: Comprehensive Vulnerability Analysis of AT Commands Within the Android Ecosystem. In *USENIX Security Symposium*, 2018.

[65] Haohuang Wen, Phillip A. Porras, Vinod Yegneswaran, and Zhiqiang Lin. Thwarting Smartphone SMS Attacks at the Radio Interface Layer. In *Symposium on Network and Distributed System Security (NDSS)*, 2023.

[66] Jinghao Zhao, Boyan Ding, Yunqi Guo, Zhaowei Tan, and Songwu Lu. SecureSIM: Rethinking Authentication and Access Control for SIM/eSIM. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2021.

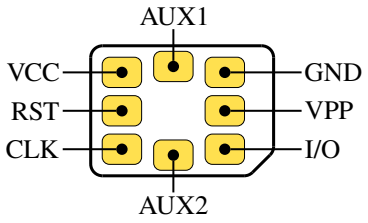

Figure 9: A 2:1 scale 4FF UICC with annotated electrical contacts [12].

A SIMs: Additional Technical Details

In the following, we provide an additional technical description of the inner workings of SIMs as supplementary material.

A.1 Transmission Protocols

ICCs can offer one or more transmission protocols by advertising them in the Answer-to-Reset (ATR). Every protocol is assigned a unique number called the protocol type T. Three most common types are as follows [26]:

- $T = 0$ offers a half-duplex transmission of characters. This means that only one side may transmit at any time, and that the smallest unit of transmitted data is a byte.
- $T = 1$ offers a half-duplex transmission of blocks. This means that only one side may transmit at any time, and that the smallest unit of transmitted data is a collection of characters, i. e., multiple bytes that form a block frame.
- $T = 15$ indicates that the ATR contains the global interface bytes, i. e., additional properties describing the parameters of the integrated circuit of the card. While not a protocol, it shows that the protocol type must not only refer to communication protocols.

SIMs support protocols $T \in \{0, 1, 15\}$, but $T = 0$ and $T = 15$ are often the only two that get implemented. We tested 10 different commercial SIMs, from different vendors and dates of production, and have not found a single card offering $T = 1$.

A.2 Communication Protocol

SIMs have one electrical contact dedicated to I/O (as shown in Figure 9). This single wire acts as the physical layer for the half-duplex data transfer protocol used for all terminal-to-SIM communications.

TPDUs are used to transfer APDUs but contrary to contemporary computer networks, layers of the network do not wrap the payload in progressively more headers to achieve encapsulation. Instead, the transmission layer introduces procedure bytes, i. e., a type of acknowledgement, that controls all data transmissions [12, 26].

All TPDUs sent to the card are mapped to C-APDUs that are structured as shown in Table 1. Field P3 of the TPDU is particularly important as it encodes fields L_c and L_e of the C-APDU, i. e., it indicates lengths of command and response payloads.

Once a TPDU is received, the ICC creates a R-APDU per Table 1 where the data is optional and the status word conveys success, errors, or warnings concerning the processing of the command [27]. The R-APDU is sent as-is, directly to the terminal (no transmission-layer interference).

The ICC receives TPDUs in one or more 2-step exchanges; it is done this way to ensure that both sides know how many bytes will be sent over the data line, and to minimize the amount of exchanged data if the card would like to reject or process a command early. In step one, the interface transmits a header to the ICC, and in step two, the ICC responds with a procedure byte. The value of the procedure byte dictates how the data exchange will proceed; the possible values are as follows [26]:

- **ACK** (= INS of the TPDU), indicates that all remaining data bytes shall be sent.
- **ACK** $\oplus$ `0xFF` (= INS $\oplus$ `0xFF`), indicates that the next (one) data byte shall be sent.
- **NULL**, indicates that no more data shall be sent.
- **SW1** followed by **SW2** completes the command.

Depending on the procedure byte sent, the interface will proceed by: Sending the remaining chunks of the command data, sending another command header, or waiting for a response. In any case, after every chunk transmitted by the interface, the ICC must respond with a procedure byte until it receives all TPDU data. Once a TPDU is received, the ICC forms an R-APDU and sends it back to the interface in a single message. Crucially, an application layer message may require multiple TPDUs to get fully transmitted, therefore the intermediate R-APDUs are considered part of the transmission layer; only the final R-APDU will be considered as a response to the C-APDU.

A.3 Commands

The communication protocol is designed such that every exchange forms a C-RP out of a C-APDU and an R-APDU. For any C-RP, the fields of the R-APDU are always interpreted in the same way, conversely, the C-APDU is interpreted in one of four ways, depending on the INS and CLA fields contained in the command header [12, 26, 27].

Every instruction, in every command class, may take a length N_c of data as input, and offer N_e data bytes as output. N_c and N_e are encoded by fields L_c and L_e of the C-APDU respectively; four possible command encodings dictate the absence or presence of L values:

Table 4: Fields of a TPDU message for protocol $T = 0$.

Command			
	Field	**Description**	**Size in Bytes**
Header	CLA	Command Class	1
	INS	Instruction	1
	P1	Parameter 1	1
	P2	Parameter 2	1
	P3	Parameter 3	1
Data		Command Data	$0 \leq n \leq 255$

Response			
		Response Data	$0 \leq N_e \leq 256$
	SW1	Status word byte one	1
	SW2	Status word byte two	1

1. L_e and L_c are absent, hence $N_e = 0$ and $N_c = 0$.
2. L_e is present and L_c is absent, hence $N_e > 0$ and $N_c = 0$.
3. L_c is present and L_e is absent, hence $N_c > 0$ and $N_e = 0$.
4. L_c and L_e are present, hence $N_c > 0$ and $N_e > 0$.

Values of N are encoded by L according to Equation 1 and Equation 2. Furthermore, $L_c = 0$ is reserved for the extended APDU, therefore it is undefined within SWICC or assumed empty ($N_c = 0$) [27].

$$N_e = \begin{cases} L_e & \text{if } 0 < L_e \leq 255 \\ 256 & \text{if } L_e = 0 \end{cases} \quad (1)$$

$$N_c = \begin{cases} L_c & \text{if } 0 < L_c \leq 255 \end{cases} \quad (2)$$

A.4 Mapping APDUs to TPDUs

A C-APDU is transmitted using one or more TPDUs, therefore, it is important to describe how each case of the C-APDU is mapped to TPDUs messages.

Case 1 requires a single TPDU header with P3 = `0x00`. No payload is present in neither the C-APDU nor R-APDU. The final status word (R-APDU) is sent in response to the first TPDU.

Case 2 requires a TPDU header with P3 = `0x00`, then depending on L_e, the card will respond with a status word that conveys the expected value of P3, or a response containing 256 bytes and a status word. In the first subcase, another TPDU is sent with P3 = L_e, and only then, the card will send the desired response.

Case 3 Transmission of a case 3 C-APDU with P3 = L_c, requires the use of the ACK procedure. The header must first be acknowledged before the command data can be transmitted to the card.

Case 4 In order to transmit a case 4 C-APDU, the command chaining mechanism is necessary. Command chaining works by first sending a case 3 C-APDU, to which the card will respond with the status word: SW1 = `0x61` and SW2 = L_e, then sending one or more case 2 C-APDUs (with INS = GET RESPONSE = `0xC0`, P1 = `0x00`, P2 = `0x00`, and P3 = L_e) that retrieve the L_e bytes from the card. This mechanism may also be used in cases where the response data is oversized, therefore does not fit in a single R-APDU [12, 27].

A.5 Java Card

SAT is an optional extension of SIMs, that allows installation of applications on the card. These applications are often packaged into cardlets. Every cardlet is constrained to a very narrow subset of the Java language where even string literals are forbidden. This is done so that cards can contain a stripped-down version of a Java Virtual Machine (JVM) for executing the application when certain APDUs are received. The COS provides standard and proprietary APIs for the application to use when interacting with card hardware and software [2, 4, 14, 19].

B SIM Interposer

Figure 10 shows a commercially available interposer for bypassing iPhone carrier locks.

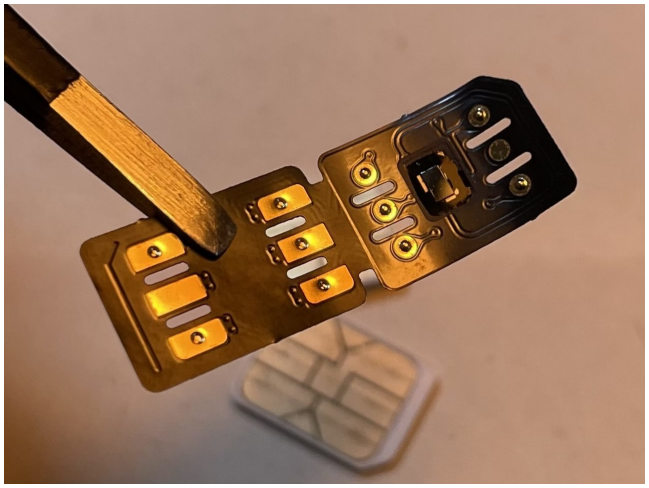

Figure 10: An interposer; a very thin, flexible smart card with contacts on both sides. It is typically placed between a SIM and the phone to intercept and rewrite communications.