A Hardware-Assisted Proof-of-Concept for Secure VoIP Clients on Untrusted Operating Systems

Maik Ender*, Gerd Düppmann†, Alexander Wild*, Thomas Pöppelmann* and Tim Güneysu*
*Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
Email: {maik.ender, alexander.wild, thomas.poeppelmann, tim.gueneysu}@rub.de
†University of Duisburg-Essen, Germany
Email: gerd.dueppmann@uni-due.de

***Abstract*—In this work we propose a secure architecture for Voice-over-IP (VoIP) that encapsulates all security and privacy critical components and I/O functions into secure hardware and thus drastically reduces the underlying trusted computing base. Our proof-of-concept implementation shows that high security and reliance on established standards and software (e.g., device drivers, transmission control, and protocols) to keep development costs down are no contradiction. Security is ensured as all security and privacy critical operations of the VoIP system are performed in protected hardware and as a consequence a successful attack on any software component (e.g., buffer overflow) does not lead to a violation of security. All I/O devices like microphones, speakers, displays, and dial buttons are directly connected to the secure hardware and cannot be controlled by an adversary even if the software part has been compromised.**

I. Introduction

Voice-over-IP (VoIP) has emerged to replace conventional telephones due to its greater flexiblity and voice quality at reduced costs. However, switching from a dedicated infrastructure (e.g., landline phones or switching boxes) for voice communication to an IP-based network introduces a large number of security and privacy issues. For example, Ang Cui [1] showed in 2012 how to exploit security vulnerabilities of the operating system in Cisco Voice-over-IP (VoIP) phones. These phones are predominantly used by governments, enterprises and research organizations in privacy and security-sensitive areas. The attacks were mainly possible as the analyzed phones are based on general-purpose hardware with an Unix-like Operating System (OS) kernel and userland software. Even though the operating system was hardened, it was possible for attackers to develop exploits and acquire administrator/root privileges. This allowed eavesdropping of ongoing communication, modification of transmitted speech data but also eavesdropping on verbal communication performed in the room the phone is located in, by silently turning on the microphone. In this case, encryption and integrity checking of data during transport provides no protection, as the adversary is able to obtain or exchange the encryption keys or the raw audio data directly on the endpoint.

A possible reason for such vulnerabilities is the requirement of minimal time-to-market and availability of powerful embedded processors (e.g., the ARM series) so that vendors are tempted to use full-blown operating systems like Linux, Android, or custom Unixes for products which are connected to the Internet. In this case, developers can directly rely on available libraries, drivers, operating system functions, and tools to realize Wi-Fi routers, printers, TVs, digital cameras, point of sale terminals, or the above mentioned VoIP telephones. However, the usage of large software stacks requires the reliable deployment of security updates. Unfortunately, not all vendors release updates due to maintenance costs and not all users install them, e.g., due to compatibility reasons. As a consequence, a large number of network-enabled embedded devices is currently running with vulnerable outdated software [2]–[5].

Contribution. In this work we are dealing with this unpleasant situation by proposing an alternative architecture to secure VoIP by an implementation that (1) encapsulates security-critical operations on reasonably cheap hardware, (2) is protected against software attacks, and (3) still allows to utilize the full power of the operating system and previously developed drivers as easy interface to the hardware. For this we have analyzed all privacy and security critical components of the VoIP system and achieved a higher level of security by moving all security-relevant components into the hardware fabric of a low-cost Xilinx Zynq platform. As all physical interfaces (e.g., microphone, display, or loudspeakers) are directly connected to the hardware components, no attacker can intercept unencrypted and security-sensitive data streams – even with full access to the operating system.

Related Work. Lightweight protection of systems and keys has been previously achieved by the deployment of a hardware-based root of trust [6]. The extension of this concepts has led to new architectures for secure systems using cryptography or hardware-based methods for protection against physical and runtime attacks [7], [8]. However, these architectures are usually not commercially available or provide only protection against certain attack vectors. Practically realized approaches for secure key storage [9], [10] or against software exploits [11], [12] usually rely on the extension of Field Programmable Gate Array (FPGA) soft-cores or small research processing platforms. On reconfigurable hardware the designer can also exploit security features build in by the FPGA vendor [13], like bitstream encryption. However, the performance of soft-cores is limited compared to hard-cores that become available with the ARM Cortex family on some FPGA devices. As auditing of large software stacks for applications in the Internet of Things (IoT) [14] or mobile computing is expensive and error-prone, concepts like TrustZone have been introduced [14]. TrustZone uses proprietary hardware features of the processor to create an authenticated and trusted execution environment for security-critical programs. However,

this still requires correct usage and implementation of all TrustZone features as the CPU and memory is shared and not physically separated. We also refer to general security requirements for VoIP which are described in standards [15] and works like [16], and [17].

Outline. This work is structured as follows: In Section II we introduce VoIP and our target hardware. In Section III we details our security assumptions and provide a high-level overview of the proposed design. Implementation details are given in Section IV and we discuss results in Section V. We finish with a conclusion and future work in Section VI.

II. Preliminaries

In this section, we introduce relevant VoIP protocols and the target device.

A. Voice-over-IP

The term Voice-over-IP (VoIP) refers to techniques that are used to establish voice calls using the IP protocol in local networks or the Internet. As almost all offices and households are connected to an IP network, usage of VoIP promises large cost savings as no separate telephone infrastructure is necessary anymore (e.g. land lines and switching equipment). Usually, for session initialization and data transmission separate protocols are used. In most implementations the Session Initiation Protocol (SIP) [18] is used for session control and call initiation and thus provides control mechanisms to create, modify or terminate a session of multiple media streams. It works in conjunction with Session Description Protocol (SDP) for identification and encapsulation of the media content. The transmission of actual media content is handled by the Real-Time Transport Protocol (RTP) or Secure Real-Time Protocol (SRTP) [19]. RTP is a widely used and flexible standard that is able to adapt media content to the available network bandwidth by compressing it with different audio and video codecs. For multimedia security, SRTP defines a profile of RTP which adds message authentication, integrity, encryption and replay attack prevention to RTP's standardized packet format for delivering media content. The most important security-related entries in a packet are the payload, a sequence number, as well as an authentication tag. The payload is the encrypted audio stream while the authentication tag is generated by a Hash Based Message Authentication Code (HMAC) over the payload. By default, SRTP supports Advanced Encryption Standard (AES) in counter mode or in f8-mode (a variant of output feedback mode of operation) for the payload encryption. For HMAC generation SHA1 is used by default [19]. As SRTP requires a symmetric key for protection of multimedia data it can be used with a pre-shared key but for scalability and usability an (authenticated) key exchange is necessary. Possible choices are Zimmermann RTP (ZRTP) [20], Session Description Protocol Security Descriptions (SDES) [21], and Multimedia Internet KEYing (MIKEY) [22] which are discussed in Section III-B.

B. Zedboard and Xilinx Zynq

For our proof-of-concept implementation of a secure VoIP client we have chosen the Xilinx Zynq 7020 that is the core of the Zedboard[1]. The Zynq 7020 is a hybrid architecture including an ARM Cortex-A9 dual-core processor and an Artix-7 FPGA in one package which are connected via internal high-speed interfaces. Generally, we refer to the ARM processor by Processing System (PS) and to the FPGA part by Programmable Logic (PL). The Zedboard provides audio Analogue Digital Converter (ADC) and Digital Analogue Converter (DAC) for audio playback and recording that are hardwired to the PL so that the FPGA has full control of the audio stream before it is forwarded to the PS. An example for an OS targeting Zynq-based devices is Xillinux[2] that implements an environment supporting a large amount of already available software, including video and audio management.

III. Design Decisions

In this section, we discuss the required security goals, our concept for secure VoIP, explain design decisions and their impact on security. Details on the low-level implementation can be found in Section IV.

A. Security Goals and Assumption

Our system should prevent social threats by clearly identifying the phone that is called or the phone of a caller. All attempts to reroute calls or impersonate phones should be prevented or be detectable by the user. As it is not possible to identify the actual user without impacting the usability, e.g., by requiring a password or smart card, we assume that phones are protected against theft and misuse. Denial of service (DOS) attacks on the service are hard to prevent and thus not taken into account for this work [16].

We further assume an attacker that has full control over the network connection and can thus freely manipulate, intercept, or retransmit the data. Moreover, we assume that the attacker is able to compromise common operating systems (e.g., Linux) and services running on them (e.g. a web server). This assumption seems realistic given the high frequency of found (and fixed) security relevant bugs in common operating systems and services[3]. The trustworthy infrastructure is a Certificate Authority (CA) which we assume to be operated securely, e.g., by using a hardware security module (HSM). We further assume that the hardware security features [13] of common FPGAs cannot be circumvented by an attacker.

B. Cryptography Protection

Protection against eavesdropping and manipulation of traffic send over the network is achieved by usage of public key cryptography and a secure CA. In general we assume a network that consists of n phones P_i for $i = 0, \ldots, n-1$. Each phone is associated with a unique identifier ID_i. Each identifier ID_i is associated with a unique phone number num_i. Each phone number num_i is associated with a public private key pair (pk_i, sk_i). Each phone P_i holds a secret key sk_i that never leaves the phone, a certificate $cert_i^{sk_{CA}}(ID_i, num_i, pk_i)$ that is signed by a CA and the public key pk_{CA} of the CA. To initialize a call, the caller P_i enters the phone number num_j of the callee. The call request is forwarded to a look-up service which maps the phone number to the Internet Protocol (IP)

[1] See http://www.zedboard.org

[2] See http://xillybus.com/xillinux

[3] See http://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33

address of the callee. The call request is forwarded to the callee and both parties exchange their certificates $cert_i$ and $cert_j$ for mutual authentication and the IDs of the involved parties are displayed. In case the verification fails, the call is denied. Otherwise, a key exchange protocol $ks = \text{EXCHANGE}(pk_i, pk_j)$ is triggered to generate a symmetric session key ks. If the call is established, the session key is used to achieve authenticity and confidentiality of exchanged audio packets by calling $\text{ENCAPSULATE}_{ks}(audio)$.

As our goal is high security we modified some of the VoIP protocols[4] to support stronger and more up-to-date cryptography with the goal of an efficient implementation on reconfigurable hardware in mind. In order to implement $\text{ENCAPSULATE}_{ks}(audio)$ using SRTP we rely on an HMAC-SHA-256 to generate message authentication codes. Secure and authenticated key establishment to implement $ks = \text{EXCHANGE}(pk_i, pk_j)$ is the most critical task that is not well supported by previously proposed protocols. As an example, ZRTP [20] does not support mutual authentication and the SDES [21] protocol requires transport layer security achieved by heavyweight and complex protocols like Transport Layer Security (TLS). The most promising preexisting standard is MIKEY [22]. However, all modes of operation of MIKEY require a timestamp which cannot be created trustworthy on reconfigurable hardware. Beside that, the implementation of forward secrecy schemes like DHKE are not mandatory in the MIKEY standard. Therefore, we decided to rely on the approach used in the NaCl library [23] for key establishment. NaCl provides an easy to use interfaces for cryptographic primitives, without any difficult setup phase to minimize implementation bugs. The used Curve25519 [24] is highly secured against real world Elliptic Curve Cryptography (ECC) attacks [25]. For signature generation NaCl used the Edwards DSA (EdDSA) algorithm. It is a variant of Digital Signature Algorithm (DSA) algorithms using Edwards curves. For the EXCHANGE function, our ephemeral elliptic curve Diffie-Hellman key exchange (DHKE) is based on *Curve25519*. To start the key establishment, party P_i generates a nonce n_i. This is the party's ephemeral secret key for the DHKE, which also ensures key freshness. Using elliptic curve cryptography, the party's ephemeral public key of the DHKE is n_iG, the nonce multiplied with the base point (G) of the curve. To ensure authenticity and integrity of the parties ephemeral public key, n_iG is signed with the users long-term secret key sk_i: $sig^{sk_i}(n_iG)$. The long-term secret key is bound to the CA signed certificate $cert_i$. With that the DHKE message consist of the ephemeral public key n_iG, the signature of $sig^{sk_i}(n_iG)$ and the certificate $cert_i$. This results in two messages that have to be exchanged between the parties P_i and P_j:

$$DHKE_i = n_iG||sig^{sk_i}(n_iG)||cert_i^{sk_{CA}}(ID_i, num_i, pk_i)$$
$$DHKE_j = n_jG||sig^{sk_j}(n_jG)||cert_j^{sk_{CA}}(ID_j, num_j, pk_j).$$

After exchanging the DHKE messages, both parties validate the signature sig^{sk} of the incoming message by using the public key pk, that is included in the certificate. If the messages validity is verified, the session key is generated by $ks = n_i(n_jG) = n_j(n_iG)$.

[4]Clearly, this design decision breaks interoperability with other implementations. However, most VoIP clients only implement basic security functions with small key sizes or are prone to man-in-the-middle attacks. Thus for a high-security solution we assume that the benefit of being able to rely more trustworthy cryptography outweighs the disadvantage of breaking interoperability.

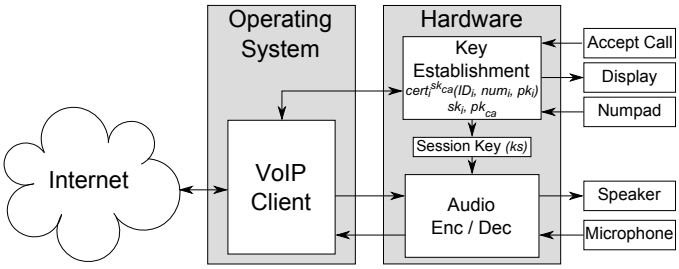

Fig. 1. Proposed VoIP architecture.

C. System Design

While the modifications of existing protocols and the concepts proposed in Section III-B are rather straightforward, the actual challenge is to securely design the system with minimal attack surface, maximum usability, and re-usage of existing components for minimal development efforts and costs. As a consequence, we propose to seal all security critical components and functions (e.g., EXCHANGE/ENCAPSULATE) into a trusted hardware based environment while the remaining VoIP functions like call control and protocol handling over the network remain in software (e.g. SIP). This results in a hardware-software co-design that ensures confidentiality and authenticity and takes into account the assumed ability of an attacker to compromise operating systems (see Section III-A).

The partitioning of the VoIP system is shown in Figure 1 where a modified VoIP client (in our case Linphone) is executed on the Xillinux operating system. The client handles signaling for incoming or outgoing calls, transmission control, and buffering of network packets. Moreover, it has been modified to simplify processing in the PL section as packets are parsed and eventually reordered in software.

The key exchange and audio data encryption and decryption are handled in the PL section which is regarded as trusted environment. The reason is that the OS is not able to access or manipulate the PL of the Zynq. As a consequence, we store and process all security credentials sk_i, $cert_i$, pk_{CA} and ks in the PL where they are inaccessible by the peripheral system and PS. Protection of the credentials is ensured by using bitstream encryption supported by the SRAM-based Artix-7 FPGA representing the PL in the Zynq system [13]. Bitstream encryption is necessary as the bitstream is part of the boot image and loaded by the boot loader before it is forwarded to the FPGA. As a consequence, an attacker might be able to extract key information from an unencrypted bitstream. The key for the bitstream decryption is set beforehand in Zynq's internal fuses that are only readable by the bitstream decryption unit of the FPGA. This prevents manipulation and usage of a stolen bitstream on a different device to impersonate a phone. As shown in Figure 1, all VoIP-relevant I/O devices like speakers or displays are directly connected to the PL section and inaccessible by the OS running on the PS. This ensures that a trusted channel (or trusted path) with the user exists to allow verification of identities of callers or callees and secure input of telephone numbers. Additionally, no voice data ever

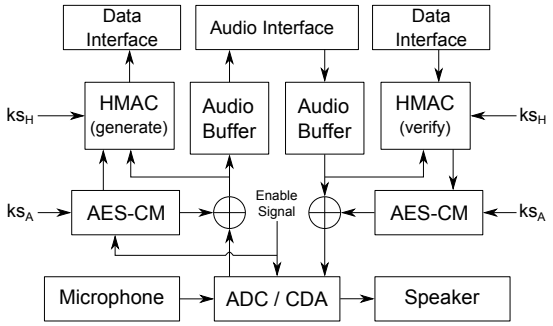

Fig. 2. The audio processing in the PL section.

leaves the PL section unencrypted which also prevents secret eavesdropping by turning on the microphone.

As previously mentioned the NaCl-based key establishment has to be implemented in the PL. In contrast to the audio encapsulation, the key establishment is based on asymmetric cryptographic primitives and complex protocols that are hard and expensive to implement in hardware. However, as it is performed only once during the setup phase of a call no real-time performance is required. As a consequence, we implemented the key establishment in a softcore (MicroBlaze) in the PL. Even though this approach does not result in a pure hardware implementation, protection and auditability of the program running on the softcore are ensured due to its minimal code size. Moreover, we do not rely on an operating system and instantiate the softcore only with internal memory (BRAM). Additionally, the communication interfaces between the PS and the softcore is deliberately limited so that buffer size restrictions are enforced by hardware which further reduces the attack surface from a potentially compromised PS.

IV. Implementation

In this section we provide details on the implementation of the secure VoIP client. We deal with audio encapsulation, implementation of the audio decryption and encryption as well as secure key exchange.

A. Audio Encapsulation

For audio transmission we partially implemented the widely used SRTP protocol in hardware (PL) as shown in Figure 2. However, we replaced the SHA-1 hash algorithm with Secure Hash Algorithm (SHA)-256 and allowed only AES-128 in counter mode (CM) for media encryption. SRTP packets contain the payload which is the audio data encrypted with AES-Counter Mode (CM), a continuous sequence number, an authentication tag which is an HMAC value over the SRTP packet and some further header values. For the payload encryption and HMAC authentication tag generation the established 256 bit session key ks is split into two 128 bit subkeys ks_A and ks_H and forwarded to the AES and HMAC cores.

1) Audio Encryption/Decryption: RTP specifies some audio compression algorithms to improve the quality of transmitted data. In the proposed system, the PS receives only encrypted audio data. As an efficient hardware implementation of audio compression is out of the scope of this work, we decided to use the simple L16 codec for this proof-of-concept [26]. L16 delivers a reasonable audio quality on uncompressed 16 bit audio samples at a variable sample rate. Our hardware implementation is equipped with two AES modules, one for encrypting outgoing and one for decrypting incoming audio streams. Both are initialized with the established session key ks_A. Even though the required throughput is low and could be handled by one AES implementation we instantiated two cores to keep a clear structure and isolate incoming and outgoing audio traffic. The AES-CM instances generate a 128 bit keystream output per run, while the audio samples given by the ADC or forwarded to the DAC are just 16 bit wide. Thus the generation of 128 keystream bits is just triggered by each eighth audio sample. Our design is running with $100\,\mathrm{MHz}$ clock frequency while the audio sample rate of the chosen L16 codec is set to a high quality rate of $44.1\,\mathrm{kHz}$. As a consequence, the performance requirements of the AES cores are low, as they have to be updated only approximately every $\frac{8 \cdot 100 \cdot 10^6}{44.1 \cdot 10^3} = 18140$ clock cycles. In general, User Datagram Protocol (UDP) is used to transmit SRTP packets due to latency and throughput reasons. As UDP packet can be lost during transmission the receiver or especially the AES-CM core responsible for the audio decryption must be able to deal with lost packets. Thus the counter value used during decryption by the AES core is derived from the SRTP sequence number. The FSM controlling the AES-CM core for the recorded audio stream and the ADC are connected to a hardware switch which keeps them in reset state. This switch is an additional security mechanism to ensure, that no audio data is silently transmitted to the PS. For usability, this switch can be combined with a user interaction like the raise of the phone handset.

2) Data Authenticity: To ensure data authenticity the SRTP standard uses an HMAC initialized with the session key ks_H. The HMAC value is generated over the encrypted packet payload, the sequence number, some other meta data and is stored in the header of each SRTP packet. Similar to the audio encryption, two HMAC instances are placed in our design to isolate incoming from outgoing data. As depicted in Figure 2, both HMAC instances are connected to the audio channel and the data interface to the PS. The HMAC verification is performed by recalculating the HMAC value from the incoming audio packet and meta values and by comparing it with the received HMAC value. Beside the authenticity check of the incoming audio stream, the HMAC core stores the last processed sequence number and compares it with the sequence number of an incoming packet. The HMAC accepts only increasing sequence numbers and rejects all packets with an equal and lower sequence number to prevent forgery and packet injection attacks.

The default Xilinux distribution provides different communication interfaces between the PS and the PL. These interfaces are used to transmit parts of the SRTP header from the PL to the PS and the other way around. The SRTP header transmission between the PS and PL is implemented differently for incoming and outgoing data. The reason for this is that the OS sometimes drops audio packet forwarded to the audio plug-in. For incoming data, the PS forwards the sequence number, the HMAC tag and the corresponding encrypted audio samples to the audio interface. The PL filters the specific header information from the audio stream. The

remaining SRTP header information is static for each session and is transmitted once at the beginning of a call through the data interface. With this information, the PL is able to decrypt the audio stream and packet loss caused by the PS based audio handling has no impact on the decryption process. The outgoing SRTP header is fully transmitted via a separate data interface. The PS now has to make sure to combine the SRTP header with the corresponding payload.

B. Key Establishment

For our proof-of-concept we use the NaCl library [23] executed on a Xilinx MicroBlaze to realize the authenticated key exchange (EXCHANGE). The MicroBlaze is a highly customizable 32-bit processor optimized for Xilinx FPGAs. Due to the simple and small program code of NaCl, block memory inside the PL is sufficient as main memory to store code, constants, and variables. Therefore, no external memory controller or caching unit is necessary which allows to keep all data protected on-chip.

1) MicroBlaze Modifications: The communication between the VoIP client software running on the PS and the MicroBlaze in the PL is realized via a dual port block RAM (BRAM) for which Xillinux automatically creates a device file for simple access from the PS. Seek and write operations in the device file are mapped to read or write operations on the BRAM. Prior to any calculation, all data is copied into the MicroBlaze's memory domain using the AXI interface and therefore the PS does not have direct access to data used in calculations or intermediate results. Providing the session key ks to the audio encapsulation module is done by standard MicroBlaze General-Purpose Input/Output (GPIO) ports. As the DHKE requires random input and no entropy source is available on the small MicroBlaze configuration we have instantiated a pseudo random number generator (PRNG)[5]. The PRNG core is connected via GPIO pins directly to the MicroBlaze and constantly generates new pseudo random numbers. Additionally, the Organic Light Emitting Diode (OLED) display is connected via GPIO to the MicroBlaze. This enables the MicroBlaze to present the phone identifier ID_j of the opposite party for each call. The phone identifier ID_j is bound to the phone certificate $cert_j$ which is verified by the MicroBlaze. Displaying the phone identifier allows a caller and a callee to identify the opposite phone preventing impersonation attacks.

2) MicroBlaze Software: To control and report the status of the state-machine from the PS, two words are reserverd in the BRAM. With the first word, the VoIP client writes requests to the MicroBlaze in order to trigger the key exchange. The second word is used to report the last processed state, e.g., which computation is done. The VoIP client is able to request the generation of $DHKE_i$ or the calculation of the session key ks and waits until the computation is done. The state-machine has three states.

1) *Reset*: In this initial state all messages and registers used in previous calls are cleared. This state can be reached at any time.
2) *DHKE generation*: A nonce n_i is generated and used for the Diffie-Hellman part n_iG. n_iG is signed and concatenated with its signature $sig^{sk_i}(n_iG)$ and the personal certificate $cert_i$. Finally, $DHKE_i$ is transmitted to the PS by moving it to a specific position in BRAM.
3) *DHKE verification*: An incoming message $DHKE_j$ is verified and the session key ks calculated. First, the certificate $cert_j$ and the signature $sig^{sk_j}(n_jG)$ are verified. Then the phone identifier ID_j is shown on the OLED display. At last the session key ks is generated, e.g., the final step of the Diffie-Hellman Key Exchange (DHKE) is performed and ks provided to the audio encapsulation section. Reaching this state is only possible after state (2) was successfully processed.

As mentioned, just specific parts of the VoIP client are moved to the PL. Thus, the handling of a call session, sorting of incoming audio packets and further control instances still remain in the software client. Therefore, the client only needs to forward messages provided or addressed to the state machine, which are the $DHKE$ messages. For performance reasons, the VoIP client can trigger the MicroBlaze to precompute the $DHKE_i$ and keep it in registers. During the initialization of a call, the cached $DHKE_i$ is wrapped into an SDES packet and send to the opponent party and the VoIP client waits for the incoming message $DHKE_j$. Further, this packet is forwarded to the MicroBlaze and the ks generation will be triggered.

V. Results

In this section we discuss our results which were obtained after place-and-route (PAR) synthesis on a Xilinx-XC7Z020-3 using Xilinx ISE 14.6. Table I provides the resource consumption on the FPGA with the most resource consuming parts of the design beeing the HMACs and the MicroBlaze core. The resource consumption of the MicroBlaze is rather high as the numbers also include Xillinux components like an Integrated Interchip Sound (I2S) core and communication buffers which are mostly responsible for the BRAM utilization.

TABLE I. Post-PAR resource consumption on the Xilinx-XC7Z020

Component	LUT	FF	DSP	BRAM
HMAC (incoming)	5750	3920	10	0
HMAC (outgoing)	5829	4000	10	0
AES-CM	1059	303	0	1
MicroBlaze	4932	6473	6	65
LFSR	33	32	0	0
OLED	779	450	0	1
Total	22087 (41%)	16867 (16%)	27 (12%)	66 (47%)

Beside the resource utilization, latencies and processing times of the components play an important role, especially in time critical applications like VoIP. A comparison of the obtained timings between a plain software client and the hardware/software co-design can be found in Table II. As a reference client we chose Linphone 3.6[6].

However, a fair runtime comparison between the plain software and co-design implemented key establishment is hard due to the use of different schemes and protocols. Linphone supports just SDES and ZRTP while our proposed scheme uses a certificate based DHKE. As one can see, the key

[5]For a final product this PRNG would have to be seeded by a true random number (see for example [27]).

[6]See http://www.linphone.org

TABLE II. MEASURED LATENCY OF THE DESIGNS.

Key Establishment	**Execution time (SW, Linphone 3.6)**	**Execution Time (Co-Design)**
Ephemeral Public Key	3.2ms	64.32ms
Signature Generation	1.2ms	34.6ms
Signature Verification	3.4ms	100.80ms
Session Key (Precomp. Eph. Key)	10.2ms	284.75ms
Audio Encapsulation		
AES Encryption	–	140ns
HMAC Generation	–	1700ns
Total	120µs	1840ns
Audio Decapsulation		
AES Decryption	–	140ns
HMAC Verification	–	1710ns
Total	120µs	1850ns

establishment runtime of the co-design is significantly higher. In general, the hardware supported key establishment timings can be decreased by implementing the key exchange in logic instead of using a softcore. But the timing delay of 284.75ms during the setup phase of a call seems acceptable for a first prototype implementation and can be certainly reduced by additional optimizations. Note also that besides the even faster audio encryption/decryption time, the co-design only adds minimal latency on the audio stream compared to a non-protected call. This is due to the parallel audio processing in hardware.

VI. CONCLUSION AND FUTURE WORK

In this work we have described a hardware-software co-design prototype for secure VoIP communication on the Zynq programmable SoC architecture. A central idea is to secure an interactive system by sealing all I/O operations and keys into secure hardware and using the software only for transmission and management of cryptographically protected data. Future work consists of the application of this concept to other scenarios like secure video conferencing or industrial control systems. Moreover, the implementation on other hybrid FPGA families like the Altera Cyclone V or the design of a custom board with minimal peripherals for a low-cost prototype seem worthwhile. Additionally, we plan to improve bandwidth efficiency and audio quality by implementing a more advanced audio codec in the PL. Moreover, the current implementation of NaCl on the MicroBlaze could be removed in favor of a pure hardware implementation since first results for Curve25519 seem to be promising [28].

REFERENCES

[1] A. Cui and M. Costello, "Hacking Cisco phones: Just because you are paranoid doesn't mean your phone isn't listening to everything you say," 29th Chaos Communication Congress (29C3), http://events.ccc.de/congress/2012/Fahrplan/events/5400.de.html, 2012.

[2] A. Cui, M. Costello, and S. J. Stolfo, "When firmware modifications attack: A case study of embedded exploitation," in *NDSS*. The Internet Society, 2013.

[3] A. Cui and S. J. Stolfo, "A quantitative analysis of the insecurity of embedded network devices: results of a wide-area scan," in *ACSAC*, C. Gates, M. Franz, and J. P. McDermott, Eds. ACM, 2010, pp. 97–106.

[4] J. Viega and H. Thompson, "The state of embedded-device security (spoiler alert: It's bad)," *IEEE Security & Privacy*, vol. 10, no. 5, pp. 68–70, 2012.

[5] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A large scale analysis of the security of embedded firmwares," in *USENIX Security*. USENIX Association, 2014.

[6] J. S. Dwoskin and R. B. Lee, "Hardware-rooted trust for secure key management and transient trust," in *ACM Conference on Computer and Communications Security*, 2007, pp. 389–400.

[7] G. E. Suh, D. E. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "AEGIS: architecture for tamper-evident and tamper-resistant processing," in *ICS*, 2003, pp. 160–171.

[8] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: a virtual machine-based platform for trusted computing," in *SOSP*, M. L. Scott and L. L. Peterson, Eds. ACM, 2003, pp. 193–206.

[9] L. Bossuet, M. Grand, L. Gaspar, V. Fischer, and G. Gogniat, "Architectures of flexible symmetric key crypto engines - a survey," *ACM Comput. Surv.*, vol. 45, no. 4, p. 41, 2013.

[10] L. Gaspar, V. Fischer, L. Bossuet, and M. Drutarovský, "Cryptographic extension for soft general-purpose processors with secure key management," in *FPL*. IEEE, 2011, pp. 500–505.

[11] S. Lukovic, P. Pezzino, and L. Fiorin, "Stack protection unit as a step towards securing MPSoCs," in *IPDPS Workshops*. IEEE, 2010, pp. 1–4.

[12] L. Davi, P. Koeberl, and A.-R. Sadeghi, "Hardware-assisted fine-grained control-flow integrity: Towards efficient protection of embedded systems against software exploitation," in *DAC*. ACM, 2014, pp. 1–6.

[13] S. Trimberger and J. Moore, "FPGA security: Motivations, features, and applications," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1248–1265, Aug 2014.

[14] A. Ukil, J. Sen, and S. Koilakonda, "Embedded security for Internet of Things," in *Emerging Trends and Applications in Computer Science (NCETACS), 2011 2nd National Conference on*, March 2011, pp. 1–6.

[15] Nist and E. Aroms, *NIST 800-58 Security Considerations For Voice Over IP Systems*. Paramount, CA: CreateSpace, 2012.

[16] A. D. Keromytis, "A comprehensive survey of voice over IP security research," *IEEE Communications Surveys and Tutorials*, vol. 14, no. 2, pp. 514–537, 2012.

[17] D. Endler and M. Collier, *Hacking Exposed VoIP: Voice Over IP Security Secrets & Solutions*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 2007.

[18] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "(RFC 3261): Session initiation protocol," RFC 3261 (Proposed Standard), 2002.

[19] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, "(RFC 3711): The secure real-time transport protocol," RFC 3711 (Proposed Standard), 2004.

[20] P. Zimmermann, A. Johnston, E. Avaya, J. Callas, and I. Apple, "(RFC 6189) ZRTP: Media path key agreement for unicast secure RTP," RFC 3711 (Proposed Standard), 2011.

[21] F. Andreasen, M. Baugher, D. Wing, and C. Systems, "(RFC 4568) session description protocol (SDP) security descriptions for media streams," RFC 4568 (Proposed Standard), 2006.

[22] J. Arkko, E. Carrara, F. Linholm, M. Naslund, K. Norrman, and E. Research, "(RFC 3830) MIKEY: Multimedia internet KEYing," RFC 3830 (Proposed Standard), 2004.

[23] D. J. Bernstein, T. Lange, and P. Schwabe, "The security impact of a new cryptographic library," in *LATINCRYPT*, 2012, pp. 159–176.

[24] D. J. Bernstein, "Curve25519: New Diffie-Hellman speed records," in *Public Key Cryptography*, 2006, pp. 207–228.

[25] D. J. Bernstein and T. Lange., "SafeCurves: choosing safe curves for elliptic-curve cryptography." http://safecurves.cr.yp.to, accessed 17-July-2014.

[26] A.-V. T. W. Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "(RFC 1889) real-time transport protocol (RTP): A transport protocol for real-time applications," RFC 1889 (Proposed Standard), 1996.

[27] M. Dichtl and J. D. Golic, "High-speed true random number generation with logic gates only," in *CHES*, 2007, pp. 45–62.

[28] P. Sasdrich and T. Güneysu, "Efficient elliptic-curve cryptography using Curve25519 on reconfigurable devices," in *ARC*, 2014, pp. 25–36.