FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# Real-time Encrypted Speech Communication Over Low Bandwidth Channels

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur/in**

im Rahmen des Studiums

**Computer and Network Security**

eingereicht von

**Markus Kammerstetter, Bsc.**
Matrikelnummer 0226196

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: Ao. Univ. Prof. Dr. techn. Wolfgang Kastner
Mitwirkung: Dipl. Ing. Mag. rer.soc.oec. Dr. techn. Christian Platzer

Wien, 11.05.2011

(Unterschrift Verfasser/in)

(Unterschrift Betreuer/in)

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 11.05.2011 _____

(Unterschrift Verfasser/in)

**Abstract**

Today, electronic speech-telephony and -communication is broadly available through a number of different media and technologies, where the general tendency is still directed towards digital mobile communication. Depending on the communication requirements, a user may employ a wired land line, a satellite or cell phone link or even a wireless radio channel. However, even though billions of users use these systems every day, the technologies employed may have security vulnerabilities allowing potential attackers to eavesdrop on the conversation, impersonate users or mount other attacks. Especially the growing amount of wireless technologies makes access to the communication media easier for adversaries and less likely to get caught. Hence to protect their users, it is of utter importance that these communication systems have a robust design involving strong cryptography to ensure information security properties like confidentiality, integrity or authenticity. In the scope of this thesis, we show that these security properties do not or only partly hold for the world's most widely used communication networks such as the Global System for Mobile Communications (GSM) or the Universal Mobile Telecommunications System (UMTS). While the design of GSM strictly follows a questionable security-by-obscurity approach that offers almost no protection today, even the more openly designed successor UMTS suffers from security implications. If users need to a have private conversations over those media, they usually need to use secure phone products that offer end-to-end encryption between the involved parties. However, as these systems are often based on closed designs, the users have to trust these products to be secure. We show that there are bogus products on the market that bring no significant security improvements, resulting in users who merely think that their conversations are secure, although in reality they are not. Also, commercial systems often target only a single communication medium (such as GSM) so that for each medium a user needs to employ a different voice privacy system. To tackle these issues, we developed a new and versatile stand-alone system that has a number of advantages over existing secure telephony products. The working prototypes comprise of customly designed embedded hard- and software that can be connected to a broad range of communication media. One of the key features is the ultra low bandwidth requirement of $9600$ baud and below, enabling its use over very low bandwidth channels. We based our design entirely on standardized, established and widely-used cryptographic primitives providing a high amount of security. In fact, our system does not only allow secure conversations with the information security properties confidentiality, integrity and authenticity, but it also provides Perfect Forward Secrecy (PFS), repudiation and a certain degree of plausible deniability.

## Kurzfassung

Elektronische Sprach-Telefonie und -Kommunikation sind heute weit verbreitet durch eine Vielzahl von unterschiedlichen Medien und Technologien, wobei der Trend nach wie vor in Richtung digitale Mobilkommunikation geht. Je nach Kommunikationsanforderungen kann ein Nutzer eine drahtgebundene Telefonleitung, eine Satelliten- oder Mobilfunk-Verbindung oder auch eine drahtlose Funkverbindung verwenden. Obwohl diese Systeme täglich von Milliarden von Nutzern genutzt werden, können die zugrunde liegenden Technologien Schwachstellen enthalten, die es potenziellen Angreifern erlauben, Gespräche abzuhören, sich als andere Nutzer auszugeben oder andere Angriffe durchzuführen. Besonders die steigende Zahl von drahtlosen Technologien erlaubt den einfacheren Zugriff auf die Kommunikationsmedien durch Angreifer und mindert zugleich auch die Wahrscheinlichkeit entdeckt zu werden. Um die Nutzer zu schützen, ist es besonders wichtig, dass diese Systeme ein robustes Design mit starker Kryptographie besitzen, sodass Informationssicherheits-Eigenschaften wie die Vertraulichkeit, Integrität oder Authenzität sichergestellt werden können. Im Rahmen dieser Arbeit konnten wir zeigen, dass diese Sicherheits-Eigenschaften für einige der weltweit am meisten verbreiteten Kommunikationsnetze wie das Global System for Mobile Communications (GSM) oder das Universal Mobile Telecommunications System (UMTS) nicht oder nur zum Teil gelten. Während das Design von GSM einem sehr fragwürdigen "security-by-obscurity" Ansatz folgte und heute nahezu keine Sicherheit mehr bietet, leidet auch der offenere Nachfolger UMTS unter Sicherheitsmängeln. Wenn Nutzer über diese Medien sichere Gespräche führen wollen, können sie üblicherweise auf sichere Telefonieprodukte zurück greifen, die eine End-zu-End Verschlüsselung zwischen den jeweiligen Gesprächspartern ermöglichen. Da diese Produkte jedoch häufig auf geschlossenen Designs basieren, muss der Benutzer darauf vertrauen, dass die jeweilige Implementation auch sicher ist. Tatsächlich konnten wir zeigen, dass es fragwürdige Produkte auf dem Markt gibt, die kaum einen merklichen Sicherheitsgewinn erzielen. Dies führt zu Benutzern die sich in falscher Sicherheit wiegen und zwar glauben sicher zu sein, dies aber tatsächlich nicht sind. Desweiteren zielen kommerzielle Systeme oft lediglich auf ein einziges Kommunikation-Medium (wie etwa GSM) ab, sodass ein Benutzer für jedes einzelne Medium ein anderes Sprachverschlüsselungs-System benötigen würde. Um dieser Problematik entgegen zu wirken, entwickelten wir ein vielfältiges sowie eigenständiges System, dass zu einer Vielzahl von Medien verbunden werden kann. Einer der wesentlichen Vorteile sind die ultra-niedrigen Anforderungen an die Kanalbandbreite von 9600 Baud und weniger, wodurch auch der Nutzen über stark bandbreitenlimitierte Kanäle gewährleistet werden kann. Unser Design besteht ausschließlich aus standardisierten, bewährten und weitverbreiteten kryptographischen Prinzipien, womit ein hoher Sicherheitsstandard geboten wird. Das System ermöglicht sichere Gespräche nicht nur mit den Informationssicherheits-Eigenschaften Vertraulichkeit, Integrität oder Authenzität, sondern bietet ebenso Perfect Forward Secrecy (PFS), Leugbarkeit und limitierte glaubhafte Abstreitbarkeit.

# Contents

# List of Algorithms



# List of Figures

# List of Tables

CHAPTER 1

# Introduction

## 1.1 Motivation

Today, speech-telephony and -communication is possible through a number of different media, where the general tendency is still directed towards digital mobile communication. Currently, with more than 2 billion users [4, 5] worldwide, the Global System for Mobile communications (GSM) currently is the biggest system. Yet, newer network generations such as the Universal Mobile Telecommunications System (UMTS) have emerged, that will supersede GSM in the near future. In addition to these widely established networks, numerous other possibilities for speech communication such as the analog Plain Old Telephone Service (POTS), satellite telephony, wireless radio channels or Voice-over-IP (VoIP) systems exist.
Unfortunately, most of these systems do not implement trustworthy voice security out of the box. Either they have no security protections at all (such as POTS) or those in place are inadequate. One such example is GSM, where voice is encrypted only on the Over-The-Air (OTA) interface between the Mobile Station (MS) and the Base Transceiver Station (BTS). However, due to many other inherent flaws [5] the security of GSM can be considered completely broken today. Newer network generations such as 3G are supposed to fix these security issues, but attacks start to arise as well. One such example is the class of *sandwich attacks* on the KASUMI encryption algorithm recently discovered by Dunkelman at al. [6]. Even if there would be no security flaws in a system, it would not be trustworthy if there is no end-to-end security in place, as there is always the possibility of an attacker in the middle. In fact, for widely used telecommunication networks it is mandated by law (at least in the European Union and the U.S.) that such networks can be lawfully intercepted for "the protection of national interest" [7].
At the same time this also opens the door for illicit use of lawful interception tools such as in the Greek telephone tapping case [8]. Therefore, if two individuals (we assume that they are named Alice and Bob) wish to talk securely over one of these networks, a trustworthy end-to-end security solution with a given set of information security properties is required. These primary security properties are *Secrecy*, *Authenticity* and *Integrity*. In addition, a system could also have the secondary security properties *Perfect Forward Secrecy (PFS)* and, if possible, *plausible*

*deniability*. For cellular networks, there are commercial secure phone products [9] or smart phone applications [10] that implement at least some of these properties, while other ones [11] only provide questionable security protection by strictly following a security-by-obscurity approach. However, so far there seem to be no readily available systems that implement end-to-end voice security in a generic enough way, that different underlying communication media can be utilized with the same equipment. At the moment, a *separate* and *different* voice security solution needs to be used for each network. To fill this gap, in this thesis we developed a generic end-to-end voice security system that implements all of the above mentioned information security properties while at the same time not restricting its use to a specific communication medium.
This is achieved in two steps: First, the required minimum bandwidth is constrained to as little as 4800 baud so that the system can be used even over very low bandwidth channels. This is possible by utilizing special speech coding algorithms in combination with an optimized protocol design.
Secondly, the system leverages custom designed embedded hardware that offers different ways to connect to existing communication devices such as cell phones or digital wireless radios. In order to gain a trustworthy security solution, we employ only cryptographic principles and protocols that are proven to be secure, well established and widely used. An additional flexibility gain is accomplished by running the Linux operating system on the embedded hardware. This allows to use the functionality of the Linux kernel and the software applications together with the software of our voice security implementation.

## 1.2 State of the Art

There are currently a number of commercial secure telephone products and projects. The GSMK Gesellschaft für sichere Mobile Kommunikation mbH [9] sells secure telephony products that include their own encryption engine implementation [12]. In contrast to other products, the source code of the encryption engine is available to its users, allowing them to make code reviews to ensure that the system does not include back doors. In contrast, products like the Secure Phone Miser [11] are strictly based on proprietary solutions and only very little is known on their security. Whispersys has developed a software tool named “Redphone” [10] that is available for the Android Smart Phone platform. The implementation uses the ZRTP protocol [13] developed by Phil Zimmerman et al. In addition to the ZRTP protocol, Zimmerman et al. also implemented ZFone [14], a software VoIP application utilizing the ZRTP protocol for secure communication. In contrast to the mentioned products, our system employs a modified OTR [15] key exchange and a bandwidth optimized communication protocol. Due to its key exchange it is closely related to OTR [15]. For a comparison to existing scientific projects, we would like to forward to the related work section 7.

## 1.3 Methodology

To be able to design our secure communication system, we evaluated the threats and attacks on prevalent communication networks like GSM and UMTS. We classified the attacks in active, semi-active and passive attacks. Based on this information, we examined both commercial as

well as academic end-to-end speech encryption solutions, that should solve these security issues. However, to be able to determine whether a system is secure, we had to define a communication model allowing us to derive a number of required information security properties. The target system has to meet the minimum set of our information security properties to be considered secure. After the analysis of the state of the art situation, we set out to evaluate established cryptographic principles and protocols in order to determine their usability for low bandwidth application. By using this information as foundation, we were able to design a cryptographic security system with provable security features. In the following, we estimated the hardware requirements and started to implement an embedded system having the functionality to capture and playback audio, to compress and decompress speech down to low bit rates as well as to run the Linux operating system. Once the hardware implementation was complete, we evaluated which type of software and programming libraries we could use to implement the security system, that we designed earlier. As soon as the overall implementation was ready, we created a theoretical model to evaluate the required bandwidth and latencies of the system. In addition, we carried out practical measurements which allowed us to verify our theoretic model. Finally, we performed an analysis regarding related scientific projects, which showed that our system and its properties are a novelty to the field of low bandwidth speech communication.

## 1.4 Outline of this Thesis

In Chapter 2, we show what type of security features existing GSM and UMTS networks have and how they are implemented. Based on this information, we analyze the security of these systems with regard to the information security properties *confidentiality*, *integrity* and *authenticity*. After presenting practical attacks in detail, we conclude that the examined networks do not offer adequate protections for secure conversations. In Chapter 3, we evaluate which established cryptographic principles and protocols are considered to be secure and how they can be applied for low bandwidth application. We come to the conclusion, that Elliptic Curve Cryptography is especially suited for our application, as it allows to significantly reduce the key sizes without compromising security. In Chapter 4, we define all the security properties our system should have. Based on the knowledge of the previous chapter, we design a security system that implements all of the defined security properties. In Chapter 5, we show how to implement the system in hard- and software. The resulting prototype system comprises of audio processing and compression hardware as well as a powerful ARM9 controller, capable of running Linux. On the software side, we make adaptations to low level software (such as boot loaders and the Linux kernel), but we also implement essential application code necessary for encrypted speech communication. In Chapter 6, we create a theoretical model, that can be used to calculate the resulting system latency and the required bandwidth. We evaluate the duration of the key exchange as well as the latency and bandwidth requirements in comparison to our theoretical model. In Chapter 7, we finally provide an overview of related work, whereas in Chapter 8, we present the conclusion and further work.

## 1.5 Contribution

In Chapter 2, we contribute a summary of the current state of the art for mobile network security with focus on GSM and UMTS. To tackle the security problems of these networks, we designed a novel embedded system comprising custom hard- and software. In addition, we designed a system applying security properties to audio streams, that are, to the best of our knowledge, new to the field of low-bandwidth speech communication. To prove the effectiveness of the system, we conducted an evaluation with regard to the duration of the key exchange, the required bandwidth and the resulting latency. Finally, we contribute a novel protocol design, allowing future system to incorporate the high security of our system.

CHAPTER 2

# Voice communication security on mobile networks

In this section we provide an overview of the most notable voice security solutions on cellular networks. For each type of network we take *Alice* and *Bob* as an example. They would like to have a private conversation that is protected in terms of *confidentiality*, *integrity* and *authenticity*. We define these properties as follows:

- *Confidentiality:* Alice and Bob want to be sure that no one else can read their messages.
- *Integrity:* Besides Alice and Bob no one should be able to change the content of a message without notice.
- *Authenticity:* Alice and Bob need proof that exchanged messages originated from each other.

Furthermore, we make the assumption that an either passive or active attacker named *Eve* might be present who tries to violate these security properties.

## 2.1 GSM

The Global System for Mobile communications is a fully digital radio network which has been designed with security in mind. That is, it should be "at least as secure as the wire-line system" [4]. Among the security features are *confidentiality* and *anonymity* on the radio path as well as strong client *authentication*.

### Architecture

Fig. 2.1 shows the simplified GSM architecture. The mobile station (MS) is a commodity item that has a unique identity in form of the 15-digit hierarchical International Mobile Equipment Identity (IMEI) number [16, 17, 18].

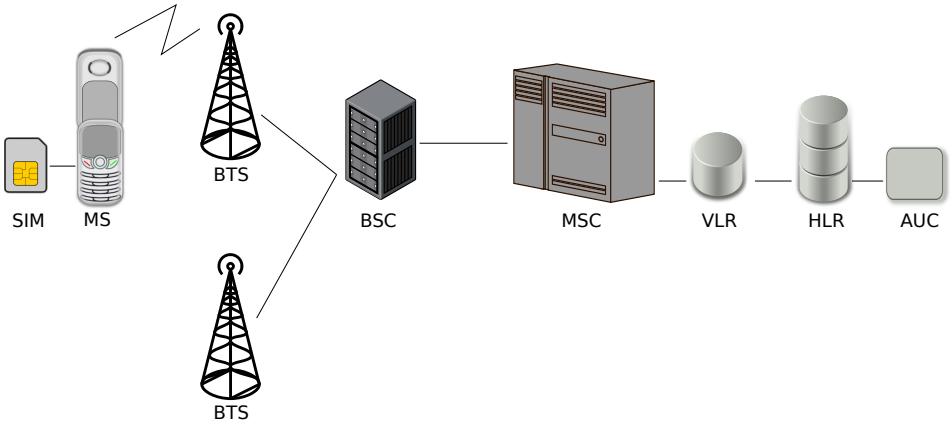


Figure 2.1: simplified GSM architecture

$$\text{IMEI}=\underbrace{AAAAAAAA}_{\text{TAC}}\underbrace{BBBBBB}_{\text{SNR}}\underbrace{C}_{\text{C/S}}$$

It is a decimal number made up of a centrally assigned 8-digit Type Approval Code (TAC), followed by a 6-digit Serial Number (SNR) assigned by the manufacturer and one check or spare digit (C/S) [18].

In order to connect to a GSM network, each mobile station needs to be personalized with a removable SIM (subscriber identity module) card. SIM cards can be native smart cards, but today mostly smart cards running the security certified Javacard [19] operating system are in use. In contrast to native implementations, this has some benefits like the GSM operators' ability to include customized applications or modifications on the card. It also allows the implementation of the required SIM functionality as hardware independent and interoperable software application. Among other data, the SIM card contains the subscriber authentication key $K_i$, the unique International Mobile Subscriber Identity (IMSI), the Temporary Mobile Subscriber Identity (TMSI) and the cryptographic authentication and key generation algorithms A3 and A8 [4, 20]. The subscriber authentication key $K_i$ is a 128-bit randomly generated secret key shared with the network provider. When the network operator issues a new SIM card, $K_i$ is stored on the SIM card as well as in the operator's Home Locator Register (HLR) database. For authentication purposes, the HLR database contains the IMSI number of the SIM card as well.

$$\text{IMSI}=\underbrace{AAA}_{\text{MCC}}\underbrace{BB}_{\text{MNC}}\underbrace{CCCCCCCCCC}_{\text{MSIN}}$$

The number is at most 15 decimal digits long and, similar to the IMEI number. It follows a hierarchical numbering concept: The first three digits of the IMSI are the internationally standardized Mobile Country Code (MCC) followed by a two digit Mobile Network Code (MNC), which uniquely identifies the mobile network within a country [17]. The remaining 10 digits (or less) denote the the Mobile Subscriber Identification Number (MSIN) applied by the home network operator to identify the subscriber.

As soon as the mobile station is turned on, the Personal Identification Number (PIN) needs to be entered to unlock the SIM card. On success, the card allows access to the IMSI number [4] and the MS connects to the strongest Base Transceiver Station (BTS). Then both the BTS and the MS start a continuous measurement process involving measurements of the received signal level (RXLEV) and the received signal quality (RXQUAL), which is a function of the number of received bit errors [17]. If the link quality is good enough, the BTS and the MS will lower their transmission power to a minimum level. This allows not only the power consumption of the MS to be minimized, but is also thwarts unwanted co-channel interference.
In the next step the MS begins the client authentication process by transmitting its IMSI to the connected BTS [4]. The message is passed on to the Base Station Controller (BSC) and the Mobile Services Switching Center (MSC), where the Visitor Location Register (VLR) and the HLR databases can be queried. Typically, a GSM network only has one central HLR, but multiple VLRs (one for each MSC) [17]. Each VLR can act as a cache for data stored in the HLR which reduces load. Besides the VLR assigns the Temporary Mobile Subscriber Identity (TMSI) which is used instead of the IMSI in subsequent network communications on the radio link. This way *Anonymity* on the radio path is achieved.

### Authentication and Encryption

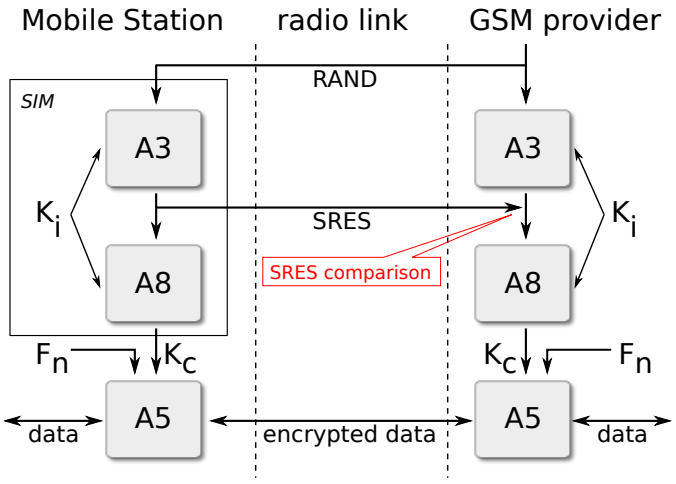


Figure 2.2: authentication and encryption in the GSM network

Fig. 2.2 outlines the authentication and encryption process. As soon as the IMSI was received by the MSC, the Authentication Center (AUC) looks up the corresponding subscriber authentication key $K_i$ in the HLR. On success, it creates five authentication triples (AUTH), but only the first triple is used while the latter ones can be applied for re-authentication at a later point in time. Each triple consists of:

$$\boxed{AUTH = RAND, SRES, K_c}$$

RAND is a 128-bit random number that is sent from the GSM network to the MS in plaintext [20]. On both sides it is fed into the A3 keyed hashing algorithm together with $K_i$ to obtain the 32-bit *signed result* SRES. The MS sends back the result to the GSM network where it can be compared with the value calculated at the network provider side. If it matches, authentication was successful and the encryption key material can be generated. In general, this kind of procedure is known as *challenge-response protocol*. On each side key material is derived by hashing SRES together with $K_i$ with the help of the A8 keyed hashing algorithm. The output of this calculation is the 64-bit encryption key $K_c$. With the calculated key $K_c$ and the current 22-bit TDMA (Time Division Multiple Access) frame number of the received transmission $F_n$, the A5 encryption algorithm is initialized and ready to encrypt subsequent messages exchanged between the MS and the BTS.

The involved cryptographic algorithms work as follows: Both, the A3 authentication algorithm and the A8 session key generation algorithm, are not standardized which allows GSM operators to choose their own implementations. Yet a non-public proprietary reference implementation of the so called *COMP128* algorithm was suggested by ETSI (European Telecommunications Standards Institute) to GSM operators [4, 21]. It combines the functionality of A3 and A8 which is why it is also known as the “A3/A8” algorithm. By reason of the non-public design, the algorithm

did not undergo public scrutiny before deployment. But due to a leaked document and reverse engineering of a handset, the algorithm became public in 1997 [21].

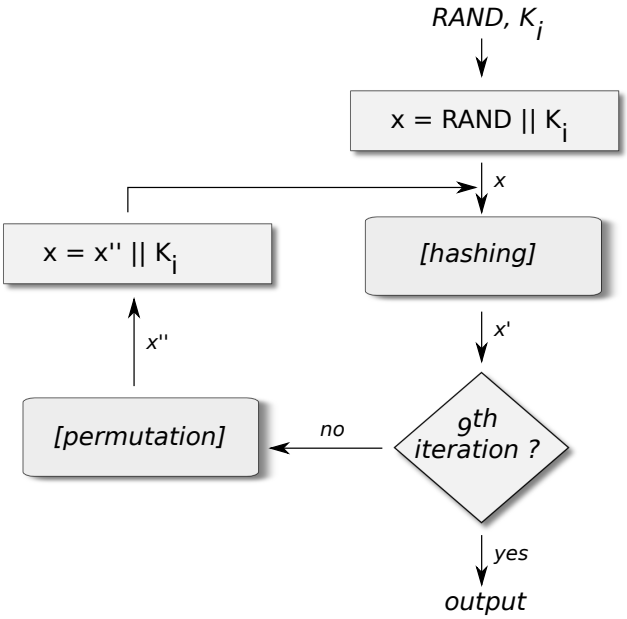


Figure 2.3: COMP128 architecture

The architecture of COMP128 is depicted in Fig. 2.3. In brief, COMP128 is initialized with the concatenation of the 128-bit RAND and the 128-bit authentication key $K_i$ [22]. The core of the algorithm consists of a hash function that transforms a 32-byte value $x$ into a 16-byte hash value $x'$ by means of compression and substitution with a set of 5 S-Boxes. The result of the hashing function $x'$ is permutated and concatenated with $K_i$ to get a 32-bit value $x''$ that is fed back into the hash function for the next round. In total 9 such rounds (i.e. hashing followed by permutation) are executed while in the last round no permutation is performed. From the output, bits 0 to 31 are used as SRES and bits 74 to 127 are applied to generate $K_c$. This is surprising, as $K_c$ does not use a full of 64 bits of the hash output, but merely 54 bits. In order to get the 64-bit key $K_c$, the latter 10 bits are filled with zeros. This effectively reduces the size of the session key $K_c$ to 54 bits [22]. For encryption there are different variations of the A5 algorithm: the stronger A5/1 variant and the weaker A5/2 variant which was designed for export use. For performance reasons they are not implemented on the SIM card, but reside on the MS instead. Like COMP128 they were designed in secret and could not undergo public scrutiny, but at some point they leaked and became public [4, 22]. In fact the non-public nature of the algorithms is a violation of Kerckhoffs' principle that the security of a cryptosystem should solely depend on the key and not on the secrecy of any other part of the system. Therefore, the design can be described as *security-by-obscurity* design. Besides A5/1 and A5/2, there is the *dummy* encryption algorithm A5/0, which does not encrypt at all, and the recent KASUMI (A5/3) algorithm which is available on 3G handsets. We will discuss KASUMI in Section 2.2.

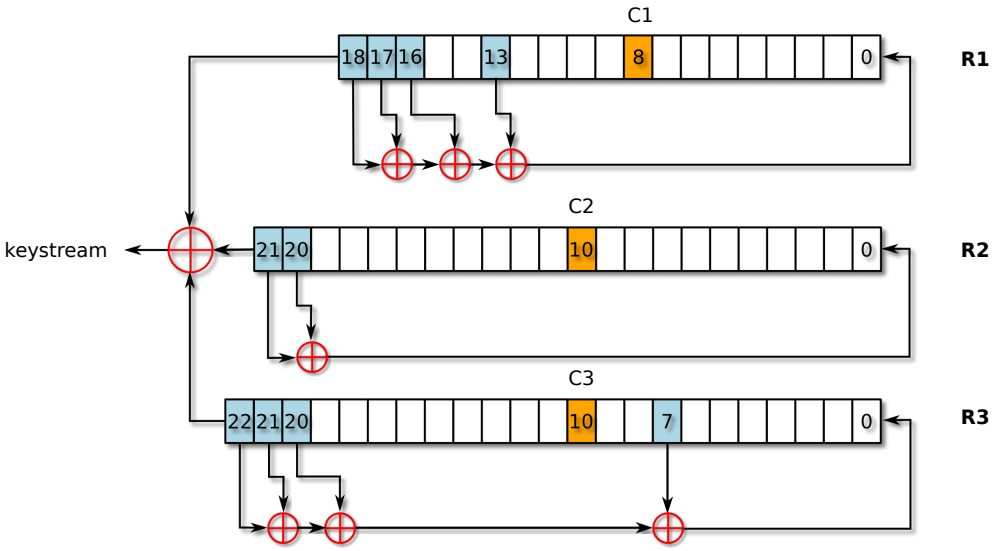


Figure 2.4: A5/1 algorithm [23]

Fig. 2.4 shows the stronger A5/1 type of the A5 stream cipher. It is implemented as a set of three linear feedback shift registers (R1, R2, R3). In each cycle a register $Rn$ is clocked, if its clocking bit $Cn$ agrees with the majority of the clocking bits (C1, C2, C3), where the majority is defined as $maj(a, b, c) = ab \oplus ac \oplus bc$. Each register has a number of tapped bits that are combined with the $\oplus$ operation and fed back into the register as LSB. Similarly, the MSBs of the registers are combined with the $\oplus$ operation to generate the key-stream. At startup all three registers are initialized with zero. Then the 64-bit secret session key $K_c$ and the 22-bit frame number $F_n$ are added to the registers through the feedback path. However, in contrast to normal operation, the registers are clocked *regularly* during initialization (thus clocking with the majority rule does not apply here). After $64 + 22 = 86$ regular clock cycles the initial state $S_i$ is reached. At this point a *warm-up* phase is performed in which the generator is irregularly clocked 100 times and the output is discarded. As GSM transmissions are made up of so called *bursts*, each one carrying 114 bits of information per direction, the A5/1 algorithm is used to generate exactly $2 * 114$ bits of key-stream. The first 114 bits are applied for uplink traffic encryption while the latter ones are used for downlink decryption. (For traffic encryption and decryption the $\oplus$ operation is applied by combining each bit of data with the corresponding bit of the generated key-stream.)
The weaker flavor of A5/1 is called A5/2 and depicted in Fig. 2.5. In comparison to A5/1 (Fig. 2.4) it stands out that the tapping bits in the feedback path of A5/2 are equivalent to A5/1. However, the irregular clocking and the key-stream generation is different. Clocking solely depends on the state of the R4 register [24, 25]. Register R1 is being clocked if C1 agrees with the majority of the clocking bits C1, C2 and C3. R2 is clocked if and only if C2 agrees with the majority while R3 is clocked if and only if C3 agrees with the majority, respectively. As soon as this clocking procedure was performed on the registers R1, R2 and R3, the clocking register R4 is clocked as well. The majority function plays an important role for key-stream generation too. As shown in Fig. 2.5, for each register R1, R2 and R3, a triple of three tapping bits is fed into the majority

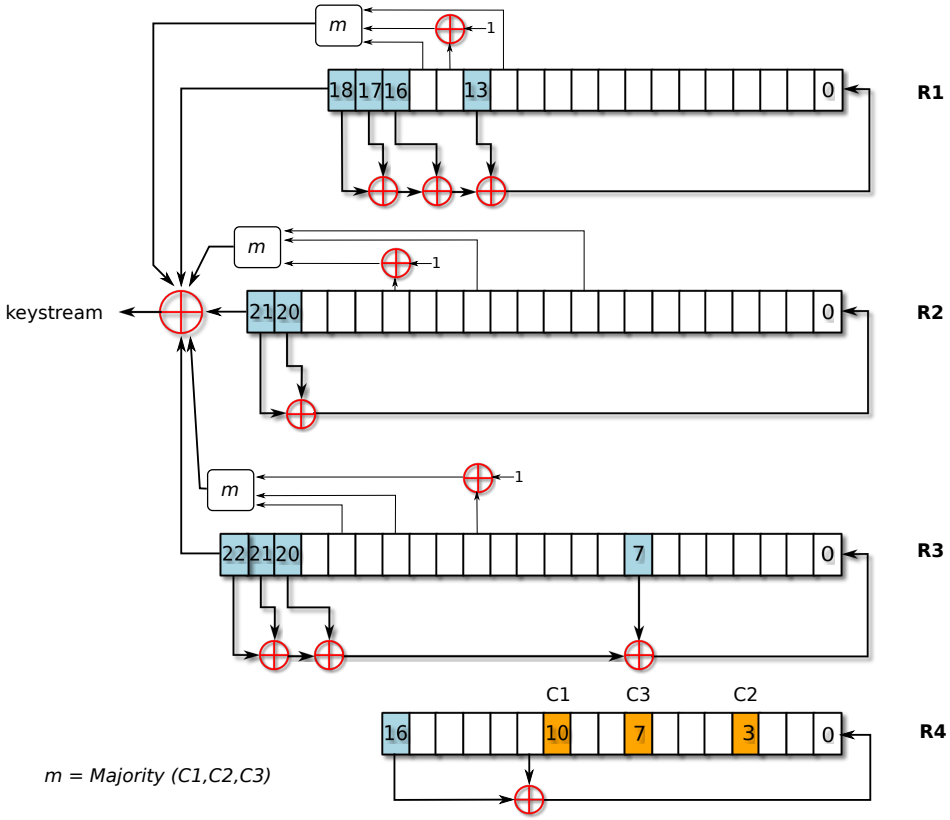


Figure 2.5: A5/2 algorithm

function $m$, where one bit of each is negated before entering the function. The key-stream is obtained by combining the outputs of these functions with the MSBs of the registers R1, R2 and R3 by utilizing the $\oplus$ operation. Initialization is similar to A5/1. First all registers are set to zero. Then the 64-bit secret session key $K_c$ and the 22-bit frame number $F_n$ are added to all registers through the feedback path [25]. At this time the registers are clocked regularly and the clocking logic does not apply. In the following warm-up phase R4 is clocked 99 times and the output is discarded. At this time the registers R1, R2 and R3 are clocked *irregularly* already. After warm-up the algorithm produces $2 * 114$ bits of key-stream, where the first 114 bits are used for encrypting the uplink and the latter ones are used for decrypting the downlink.

### Security

In this section we analyze the security of the GSM network with respect to Alice and Bob who wish to communicate securely (i.e. the properties *confidentiality*, *integrity* and *authenticity* must hold). Security issues of the GSM network itself, that do not affect the given set of properties in the communication between Alice and Bob, are less relevant to us.
As mentioned in Section 2.1, the design goals of GSM were that confidentiality and anonymity are implemented *on the radio path* only (i.e. between MS and BTS). In addition the network

only performs *client-* authentication by means of the A3/A8 algorithms (see Section 2.1), but no authentication of the network-side.
Unfortunately, for Alice and Bob this means that they can not use the GSM network to have a secure conversation with each other. They can not be sure that their call is confidential as *per design* [4, 20] there is no end-to-end encryption between their mobile stations (MS). Likewise, there is no integrity protection that would alert them if someone tries to alter the content of their conversation. In fact they can not even be sure that they are really talking to each other since their handsets do not perform mutual authentication. Instead they only authenticate to the GSM operator (i.e. to the operator's HLR).
If an attacker plans to intercept their phone call, there are different ways to do so depending on how much access she has to the GSM network. For example, a powerful adversary might be an inside attacker working at the GSM operator with direct access to the GSM network (like a technician). Due to the CALEA (Communications Assistance for Law Enforcement Act) wiretapping law in the United States and a comparable law in the European Union [7], at least in these countries telecommunication operators are mandated by law to allow "access to the entire telecommunications" [7] to Law Enforcement Agencies (LEAs) over standardized "lawful" interception (LI) handover interfaces. Unfortunately, these interception interfaces may also be subject to illicit use (such as in the Greek wiretapping scandal [4, 8, 26] in 2004).
In the above mentioned cases the security of the call between Alice and Bob can be breached without the need to overcome the GSM encryption or authentication barriers. Yet in practice it is more likely that an attacker targets the wireless communication interfaces as they are easier to access and the chances of being caught are low.
Before discussing the security of the wireless connection between the mobile station and the BTS, it is noteworthy to mention that "in most countries the communication between the base stations and the VLR pass unencrypted microwave links" [4]. If a GSM operator decides to extend the network coverage by installing new base stations, he is often challenged with the fact that there is no (cost-effective) way to connect the BTS to the underlying GSM network infrastructure (which is especially the case in rural areas). This is the reason why in such cases network operators tend to use directional microwave links to hook up their BTS to the rest of the network. Often this equipment can encrypt traffic, but "the average phone company has no incentive to switch the cryptography on" [4] which is why most GSM traffic passes in clear over the microwave links [27, 28]. In fact these links are a viable mean for intelligence agencies to "get warrant-less access to traffic" [4]. Equally, these links might be intercepted by other attackers, too.

**The wireless connection** between a mobile station and a BTS is authenticated and encrypted as described in Section 2.1. In the past intercepting this connection was considered to be very cost-intensive as specialized communication- and signal processing equipment was required [29]. An example of such devices are the so called "IMSI-catchers" that can not only receive GSM traffic, but they can also transmit (a more detailed description is available below). In addition to having a price tag of $200.000$ Euro or more [29, 30], such units are exclusively sold to government institutions [4, 30]. Today, IMSI-catchers allow to listen in on GSM conversations, but early variants could merely obtain the IMSI number of nearby handsets (hence the name *IMSI catcher*). They allow police forces to obtain IMSI numbers which can be used to get a warrant for tapping

phones (e.g. by using lawful interception) [4, 31], even if suspects use anonymous SIM cards. However, as handsets merely use the pseudonymous TMSI number in wireless transmissions, the IMSI catcher needs to rely on a trick. As soon as a MS connects to the IMSI catcher, it will first do this with the TMSI number. But the IMSI-catcher claims to not understand the TMSI and the handset kindly sends the cleartext IMSI [4]. In fact this is not a bug, but a *feature* that is needed for roaming and failure recovery purposes [4].
However, with today's technological advances it is feasible to implement low-cost ($< 2000$ Euro) GSM monitoring systems or even an IMSI-catcher (examples are given in [30] and [31]) by leveraging the numerous potentials of *software defined radio (SDR)* systems [31, 30] like the Universal Software Radio Peripheral (USRP) [32]. In contrast to traditional radio systems, these systems are low complexity general purpose transceivers that move most of the signal processing away from hardware towards a software implementation. Hence, all (de-)modulation and signal processing tasks are done in software which makes these systems extremely powerful. For example, if the SDR covers the according frequency range and bandwidth, the same device can act as broadcast radio receiver at one time and as IMSI catcher or garage door opener at another time without having to change the hardware. The only difference is the software (-configuration). In the context of GSM security it is an ideal tool for an attacker allowing her to send and receive GSM signals. In addition to the GNU Radio software toolkit [33] (which implements basic SDR signal processing), there are readily available SDR tools like AirProbe [34] or OpenBTS [35] that have been specifically designed to work with GSM.
In the following we show some practical attacks breaking the confidentiality of the conversation between Alice and Bob that can be conducted with the above mentioned low-cost tools. We differentiate between passive and active attacks. In a passive attack the adversary merely *monitors* the traffic between one of the conversational partners (i.e. Alice or Bob) and the BTS. In an active attack the adversary actively transmits and thus interferes with normal network operation (e.g. by setting up a fake base station).
If the A5/2 algorithm (Fig. 2.5) is used for encryption, the internal state and the session key $K_c$ can be recovered in real-time. Barkan, Biham and Keller have shown that a passive attacker merely needs to capture eight frames ("a few dozen milliseconds") [25] of encrypted data to mount a ciphertext-only attack that can recover the key in "less than a second on a personal computer" [25]. The general idea of this algebraic attack is to describe every bit of the key-stream output as a set of equations relating to the initial state of the registers R1, R2 and R3 and then guessing the initial state (out of $2^{16}$ possible states) of the R4 clocking register. As GSM applies error correction codes before encryption, the structure of the plaintext is highly redundant (for example on the GSM Slow Associated Control Channel (SACCH) each message has a fixed size of 184 bits prior to applying the error-correction code while it has a size of 456 highly structured bits afterwards). In their paper Barkan et al. show that this property can be applied to the ciphertext and in a wider sense also to the key-stream [25]. By combining the knowledge regarding the structure of the key-stream with the equations relating to the initial state of the registers, it is possible to construct a new system of equations that can be solved with Gaussian elimination [25, 24]. This procedure is repeated for every guess of R4 until the overall initial state for a given ciphertext is found. Likewise, from the initial state the session key $K_c$ can be obtained by reverse-clocking the A5/2 algorithm. Barkan et al. optimized their attack with a

time-memory tradeoff by pre-computing a set of matrices (having a size of 500MB) which took them roughly 5 hours [25]. As a result for a given set of eight SACCH ciphertext frames the key $K_c$ can be recovered in less than a second.
A similar approach is taken by Bogdanov, Eisenbarth and Rupp [24]. In contrast to Barkan et al. they directly attack the GSM speech channel which requires 16 ciphertext frames and allows them to start eavesdropping even during a call. Furthermore, they built special-purpose hardware that can recover the key in about 1 second *without* precomputation. Due to the severe attacks on A5/2, the algorithm was phased out by the GSM security working group in 2006 [25]. As a result newer GSM equipment should no longer support A5/2.

**A passive attack** on A5/1 (Fig. 2.4) is significantly harder. During the last decade A5/1 has been extensively analyzed, but most proposed attacks lacked practicability due to "strong preconditions, high computational demands and/or huge storage requirements" [36]. In 2008 Gendrullis, Novotný and Rupp implemented a practical attack on A5/1 on a special purpose FPGA (Field Programmable Gate Array) hardware device, called COPACOBANA (Cost-Optimized Parallel Code Breaker) [36]. The basic idea of their attack is to guess the complete content of registers R1 and R2 and then determine a large part of R3 from the known key-stream while the rest of the bits in R3 need to be guessed as well. To optimize their attack, they discard wrong possible choices for R3 by recognizing early contradictions in the algorithm's clocking behavior. The attack presumes 64 bits of known key-stream and can reveal the internal state of the cipher with a time complexity of 7 hours on average and 14 hours in the worst case. Known key-stream bits can be gathered easily by applying the $\oplus$ operation on captured ciphertext and known plaintext in the GSM protocol. An example of known plaintext is the *Cipher Mode Complete* message, which is the first encrypted message after the A5/1 encryption has been enabled. It usually consists of empty padding bytes [37]. For an adversary the attack by Gendrullis et al. is effectively usable by first capturing an encrypted communication with the victim and then, at a later point in time, decrypting it within the mentioned processing time.
Yet recently it has been shown by Nohl et al. [37] that a *time-memory trade-off* (TMTO) attack yields to a practical real-time key recovery. The central idea of this attack is that in theory a code book could be created that maps every secret internal state of the cipher (i.e. the state of the registers R1, R2 and R3) to a key-stream output. Hence by obtaining several bytes of known key-stream (as described above), only a single code book lookup would be required to obtain the secret internal state of the cipher and consequently also the session key $K_c$ by means of reverse-clocking. At a first glance for a 64-bit cipher such a code book would require roughly 128 Petabytes of space and it would take more than 100.000 years of computation time on a single-core CPU [37]. However, there are several key factors that allow a practical implementation of this attack:

- A5/1 can be efficiently implemented to run on graphic cards. Consequently it would be possible to create the complete code book within *3 months* on 80 state of the art graphic cards [37].
- Instead of storing the complete code book on disk, only a small part of state mappings can be stored in an optimized *rainbow table* [38]. This results in an arbitrary time-memory

trade-off: The smaller the rainbow table, the longer is the computation time necessary for the table lookup.

- As several states collide by repeatedly clocking the cipher, the warm-up phase of the cipher (see Section 2.1) effectively reduces the key space from 64 bit to merely 61 bit [37].
- The A5/1 rainbow table can be compressed efficiently.

As a result, a set of rainbow tables has been calculated which requires a total of 2TB storage space. With the rainbow table set and two captured encrypted known plaintext messages, the secret key can be recovered with 90% probability [37, 39]. If the attack is carried out on a PC with fast disk access and two state of the art graphic cards, the session key $K_c$ can be obtained within 5 seconds [37]. Today both, the rainbow tables and the associated open source cracking software *Kraken*, are public and can be downloaded from the Internet [39].

**A very practical** active attack exploits the fact that the BTS does not need to authenticate to the MS (see Section 2.1). It allows an adversary to set up his own fake BTS in the vicinity of the target which impersonates the target's genuine network by adopting the network parameters (i.e. mobile country code (MCC), mobile network code (MNC) and the provider's short name) [4, 30, 31, 37]. These parameters can be obtained easily over the air from any of the provider's BTSs. Once the fake BTS is set up, an attacker may send arbitrary messages to connected mobile stations leading to a broad range of possible attack scenarios. For example, a specially crafted GSM message could exploit a software vulnerability in the mobile station's baseband firmware, potentially resulting in arbitrary code execution. However, in the following we assume that the attack is carried out to eavesdrop on the call between Alice and Bob. In this scenario the adversary also needs to set up a (fake) MS. To the target the fake BTS will pose as genuine BTS whereas the attacker's (fake) MS is used to connect to the GSM operator's network. With this setup the adversary can perform a Man-in-the-Middle (MitM) attack on the authentication and encryption setup (Section 2.1) process. The attack can be divided into two different phases: connection establishment and the actual Man-in-the-Middle attack.
In the first phase the adversary needs to make sure that the target's MS connects to the fake BTS. Depending on the current state of the MS, different actions are required. In the simplest case the target's MS was switched off and is being turned on while the fake BTS is in operation. If the fake BTS is close to the target, the handset's measured signal quality (see Section 2.1) of the fake BTS will be higher than the signal quality of a genuine base station and the handset will readily connect to it [30].
Slightly more effort is required if the target's MS is already connected to a BTS, but there is currently no call in progress. Whenever a MS connects to a genuine BTS, it also receives a list of channel numbers for all neighboring base stations. Each channel number, the Absolute Radio Frequency Channel Number (ARFCN), specifies the exact uplink and download frequency of the corresponding BTS. Once the MS has received the list, it will only connect to these base stations (and not to the fake base station). To circumvent the problem, an attacker can either jam the BTS channel that is currently in use by the target's MS or he can overpower the weakest neighboring BTS with his fake base station [30]. When jamming, the MS looses the connection to the BTS

and it will switch to the initial frequency scanning mode. Similar to the first case, where the handset was previously turned off, it will connect to the fake BTS as soon as jamming is disabled. On the other hand overpowering the BTS with the lowest signal quality can be accomplished if the fake BTS starts to send on the same ARFCN. This way at the MS the weakest BTS in the neighbor list suddenly becomes the strongest one. As a result the MS will disconnect from its current BTS and connect to the fake BTS [30].
Similarly, jamming or overpowering a BTS can be done if the MS is active in a call, but the effect will be that the call is either dropped or not interceptable with a MitM attack on the authentication and encryption setup. If the attacker jams the BTS channel in use by the target's MS, the active call will be dropped. While this is a nuisance for the target, he will most possibly call again (or receive a call from the other party). In the meantime the attacker can apply the techniques described above to get the target's MS to connect to his fake BTS. If the attacker overpowers a neighboring BTS instead, the MS will connect to the fake BTS, but in a different way than in the previous case where no call is in progress. During a call, the MS does not change the BTS on its own, but it needs to be commanded to do so with a *handover* message sent by the BTS [16, 30]. The decision whether a handover should be performed has to be made by the GSM network. Whenever the MS is in a call, it continuously measures the signal strength of all neighboring BTSs. The results are transmitted to the network. In the event that the received signal of a neighboring BTS is stronger than for the current BTS, the network initiates the handover to the stronger BTS (i.e. the attacker's fake BTS). But since there is still a call in progress, the adversary would need to forward all communication between the genuine (weak) BTS that she is overpowering and the target's MS. Also she has no way to mount a MitM attack on the ongoing encrypted communication, as the encrypted session (and the session key $K_c$) is already established. With that in mind we believe that the benefit of message forwarding is limited and disrupting the active call is the easier and more effective attack method.
Once the target's MS is connected to the fake BTS, the BTS can force the MS to re-authenticate [25]. At that time the attacker can also retrieve required information such as the IMSI number from the target's MS. In the following we focus on the Man-in-the-Middle attack on the authentication and encryption setup.
Essentially, the adversary starts this attack by using his fake MS to connect to a genuine BTS. In the *location update* message she has to provide the so called *class mark* information which includes sensitive information such as the encryption capabilities that the MS supports [25, 31]. It gives the attacker the ability to choose either no encryption algorithm (A5/0) or only a weak one like A5/2. In the literature this attack is known as *class mark attack* [25].
Following the authentication process outlined in Section 2.1, the network will retrieve the secret key $K_i$ from the HLR and generate the RAND token. To complete authentication, the adversary forwards the RAND token to the target's MS and the SRES response to the genuine BTS. If the attacker chose to disable encryption by only allowing the A5/0 dummy encryption algorithm in the class mark information, she can easily intercept any further communication between the genuine BTS and the target's MS. Otherwise the network will start encryption with the *start ciphering* message and the attacker has the problem that she does not know the session key $K_c$. However, as the weak A5/2 ciphering algorithm was chosen by the attacker, she can easily reconstruct $K_c$ in less than a second. In fact the GSM standard specifies that the network should wait up to

12 seconds [25, 31] for a response. Considering that the previously described passive attack on A5/1 by Nohl at al. [37] takes roughly 5 seconds to complete, it seems perfectly reasonable that today also A5/1 can be used to recover $K_c$. (This could be especially relevant if the network only supports stronger ciphers like A5/1 and A5/3 (see Section 2.2)). As soon as the attacker has the session key $K_c$, she can decrypt any further communication easily. In addition we would like to stress that with the key $K_c$ and the fake BTS, the attacker can also cut off the victim's MS, hijack the call and then impersonate the victim [25].

**Finally**, we would like to outline a semi-active attack that can be used if stronger ciphers like A5/3 are in use. The attack exploits the fact that there is no key separation when using different ciphers [4, 25]. The generated session key $K_c$ for a weak cipher like A5/2 is the same as the one for a strong cipher like A5/3. To mount the attack, the adversary passively records the network authentication (i.e. the RAND token) and the encrypted communication between the target's MS and the BTS. Then, at a later point in time, he sets up a fake base station in the vicinity of the target. The target's MS will connect to the fake BTS and the attacker sends the previously captured RAND to the MS. However, in contrast to a genuine BTS, he specifies that the MS may only use a weak cipher like A5/2. Once the encrypted communication with the weak cipher is set up, the attacker can easily recover the session key $K_c$. But as the same key was used in the previously recorded conversion, the attacker can now decrypt all previously captured messages as well.

## 2.2 UMTS/3G

Today the belief that the Universal Mobile Telecommunications System (UMTS) and 3G are the same seems to widely prevalent [4]. However, 3G actually refers to a set of third generation mobile network standards released by the 3rd Generation Partnership Project (3gpp) and the 3rd Generation Partnership Project 2 (3GPP2). As a result UMTS is only *one* of the specified third generation systems and in countries such as the United States or South Korea different 3G system like CDMA2000 are more widespread [40].
In contrast to the circuit switched GSM system, UMTS comprises a packet switched architecture with high speed packet access and improved security. GSM security features that have proven to work reliably, were adopted while the shortcomings of GSM security were addressed with new security features. Among these are mutual authentication (i.e. the network also authenticates to the user now), encryption and integrity protection with publicly peer reviewed algorithms and the use of 128-bit keys.

### Architecture

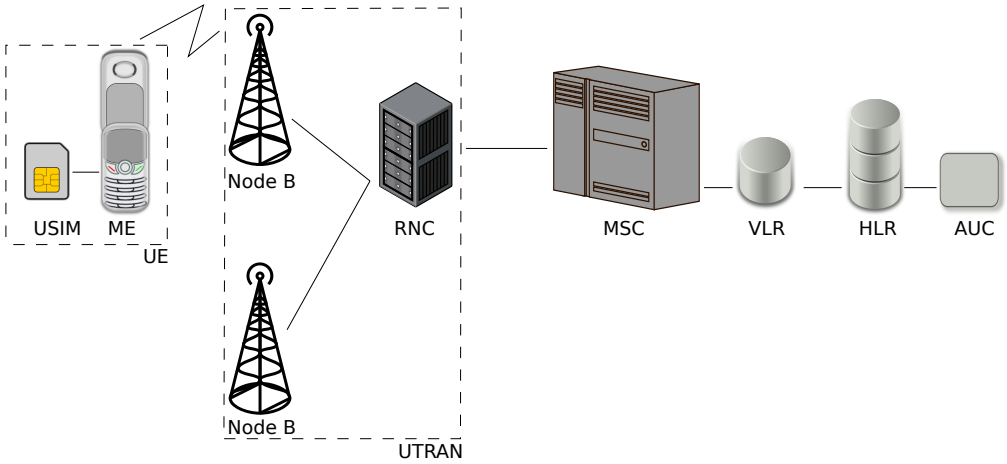


Figure 2.6: simplified UMTS architecture

Fig. 2.6 shows the simplified UMTS architecture which mostly resembles the architecture of GSM. The Universal Subscriber Identity Module (USIM) holds vital information such as the shared 128-bit subscriber authentication key $K$, the cryptographic algorithms necessary for authentication and key generation as well as the IMSI and the TMSI number [40, 41]. The USIM is a software application running on a standardized and tamper resistant smart card, the Universal Integrated Circuit Card (UICC) [41, 42]. Today, these cards are extremely powerful. For example, in addition to the USIM application they allow to run numerous (possibly value added) applications in parallel, they may have high speed interfaces such as USB 2.0 and they may even provide remote management services by running a Smart Card Web Server with IP connectivity [42]. In UMTS terminology the terminal device (e.g. a 3G cellphone) is called Mobile Equipment (ME), but as soon as it is combined with the USIM card, it is more generally

known as User Equipment (UE) [28, 40]. On the network side there are the radio base stations (Node B) which are controlled by Radio Network Controllers (RNCs). Base stations together with a RNC form the so called Universal Terrestrial Radio Access Network (UTRAN) [28, 40]. UTRANs are connected to the UMTS Core Network (CN) comprising components that we already know from the GSM architecture (Fig. 2.1). Other components that are mainly required to provide access to both circuit- and packet-switched networks have been left out from this description.

### Authentication and Encryption

The Authentication Center (AUC) and the USIM share the secret 128-bit subscriber authentication key $K$ and a set of authentication and key generation functions. In total, there are five one-way functions denoted $f1..f5$ which are in principle operator-specific, but to achieve interoperability of different USIM implementations, the 3GPP provided a set of example algorithms commonly knows as *MILENAGE* [41, 43]. The role of these functions is specified in [43] and can be seen in Tab. 2.1.

| name | role |
|---|---|
| $f1$ | network authentication function |
| $f2$ | user authentication function |
| $f3$ | cipher key derivation function |
| $f4$ | integrity key derivation function |
| $f5$ | anonymity key derivation function |

Table 2.1: authentication and key generation functions

Furthermore, to thwart replay attacks, the USIM card and the HLR keep track of two sequence number counters SQNMS and SQNHE. SQNMS denotes the highest sequence number (SEQ) the USIM has accepted while SQNHE is an individual counter for each user [44]. The MILENAGE algorithm itself is specified in [45]. The basic idea of the algorithm is to use an existing 128-bit *kernel block cipher* in different ways to compute the 128-bit output value of $f1..f5$. A simplified functional model given by Nyberg [41] is visible in Eq. 2.1, where $x$ denotes the input value, $E_K$ is the kernel encryption function with key $K$, $t$ denotes the number of distinct output blocks and the values $a_1, a_2, .., a_t$ are fixed constants. For a more detailed description, see the ETSI MILENAGE specification [45].

$$z_i = E_K(E_K(x) \oplus a_i), \text{where } i = 1, 2, ..., t \tag{2.1}$$

The specification suggests to use the well known AES-128 (Rijndael) algorithm as kernel block cipher, but also mentions that other block ciphers or keyed cryptographic functions can be used as long as they have 128-bit input, key and output values. In addition is must be computationally infeasible to determine the key from chosen input or output values and it must be infeasible to determine the output without the knowledge of the key [45].

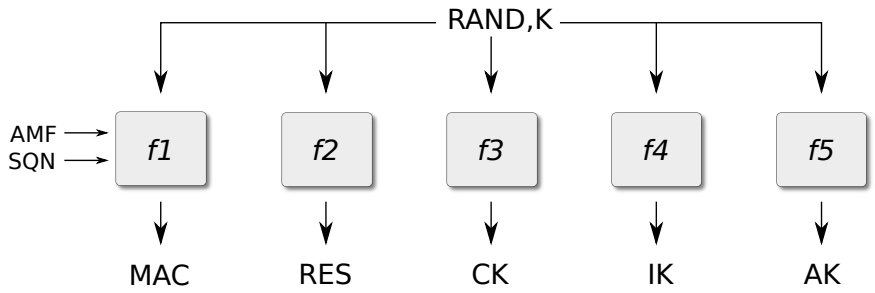


Figure 2.7: MILENAGE functions $f1..f5$ for authentication and key generation

Fig. 2.7 shows how the functions $f1..f5$ are applied to derive authentication and key material [43, 46]. This material is required to run the UMTS mutual *Authentication and Key Generation (AKA)* algorithm which we will describe in the following.
Similar to GSM, whenever UMTS User Equipment (UE) is turned on, it starts the authentication process by sending the cleartext IMSI number (see Section 2.1) to the network. If the network knows the received IMSI number, the Authentication Center (AUC) generates a 128-bit random number RAND. Together with the corresponding shared secret key $K$ the output of functions $f1..f5$ can be computed.
The function $f1$ requires the additional parameters $SQN$ and $AMF$, where $SQN$ is a fresh sequence number (with respect to the above mentioned SQNHE counter) and $AMF$ is the Authentication Management Field used "to fine tune the performance or bring a new authentication key stored in the USIM into use." [44]. The result of $f1$ is the message authentication code (MAC) for these fields.
The function $f2$ generates the user authentication output $RES$ which is applied at a later point in time for comparison in the challenge-response protocol. In principle this is the same procedure that is used for GSM user authentication.
The functions $f3..f5$ generate the cipher key ($CK$), the integrity protection key ($CK$) and the (optional) anonymity protection key ($AK$), respectively. As soon as the output of all functions is available, the AUC forms the authentication vector $AUTH$ and the quintet $Q$ visible in Eq. 2.2, where $||$ denotes concatenation. If anonymization is enabled, the $\oplus$ operation is used to conceal the sequence number $SQN$ in the authentication vector with the anonymity protection key $AK$.

$$\boxed{\begin{aligned} AUTH &= SQN \oplus [AK]||AMF||MAC \\ Q &= (RAND, RES, CK, IK, AUTH) \end{aligned}} \tag{2.2}$$

From the AUC the quintet $Q$ is forwarded to the VLR and finally to the RNC which will send $RAND$ and the authentication vector $AUTH$ to the UE [27, 46]. On receipt, the USIM first retrieves the unconcealed $SQN$ [46]. If anonymization is enabled, the USIM can unconceal $SQN$ by first deriving $AK$ with the $f5(RAND, K)$ function and then computing $SQN \oplus AK$. In the next step the MAC for the $SQN$ and the $AMF$ field is calculated with $f1$ and verified.

If the calculated MAC matches the received MAC, the USIM can assume that these fields were indeed sent by the network. (The reason for this is that in addition to the USIM, only the AUC knows the shared secret key $K$ that is required to calculate the correct MAC.) Furthermore, the $SQN$ is checked against the SQNMS counter and the counter is updated to thwart replay attacks. This way if a message with the same sequence number arrives again, the USIM will not allow it anymore and discard it. In case all verification procedures were successful, the UE calculates the user authentication token $RES$ and the keys $CK$, $IK$ with the functions $f2, .., f4$ respectively. Finally $RES$ is sent back to the network. If the received $RES$ matches with the calculated one from the quintet $Q$, the Authentication and Key Generation (AKA) process is considered to have completed successfully.

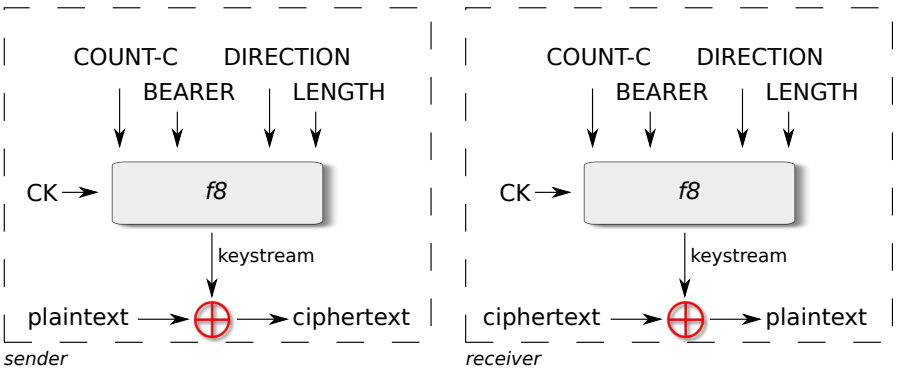


Figure 2.8: $f8$ encryption function

With the necessary keys $CK$ and $IK$, the user and the network can start to secure their communication. User packets and signaling messages are encrypted between the UE and the RNC with the encryption function $f8$ visible in Fig. 2.8 [46, 47]. The function acts as a stream cipher and produces a key-stream applied to either encrypt or decrypt a block of data by applying the $\oplus$ operation. While the main input parameters of $f8$ are the Cipher Key ($CK$) and the length (LENGTH) of the key-stream that should be generated, there are three additional parameters denoted COUNT-C, BEARER and DIRECTION [47]. These parameters are required to ensure that each frame is encrypted with a different key-stream [44]. COUNT-C is a time-dependent input (COUNT-C), BEARER is the radio bearer identity and the 1-bit DIRECTION input specifies the direction of the transmission (i.e. uplink or downlink) [40, 47]. However, the original $f8$ function has been extended to the more generic *KGCORE* function which requires the two additional parameters CA and CE. While CA specifies the mode of encryption, the CE field is currently reserved for future use. This implementational generalization allows to extend the security of the UMTS ciphers to older network generations such as GSM, EDGE (Enhanced Data rates for GSM Evolution) or GPRS (General packet radio service) with the only difference being the initialization parameter CA of the KGCORE function [44, 41]. As a result most of todays mobile phones can use the GSM network with the more secure GSM A5/3 cipher as alternative to the less secure older class of A5 algorithms.
Similar to the MILENAGE authentication functions, $f8$/KGCORE makes use of a cryptographic *kernel* algorithm. At the moment ETSI has specified the UEA1 (UMTS encryption algorithm 1)

based on the KASUMI block cipher and the UEA2 based on the SNOW 3G stream cipher [47]. For performance reasons all kernel algorithms (i.e. UEA1 and UEA2) are implemented on the UE. In addition there is the dummy algorithm UEA0 which denotes "no encryption".
The KASUMI block-cipher was designed by the ETSI Security Algorithms Group of Experts (SAGE). It has a Feistel-Network structure and is based on the MISTY1 algorithm that had already undergone public scrutiny at that time. In comparison to MISTY1, KASUMI has undergone various *enhancements* for easier hardware implementation and higher security in order to meet mobile communication requirements [41]. The close resemblance to MISTY1 is reflected in the name of the KASUMI algorithm, as *kasumi* is Japanese for "mist". For a detailed description of the KASUMI algorithm we would like to forward to the official 3GPP specification [48].

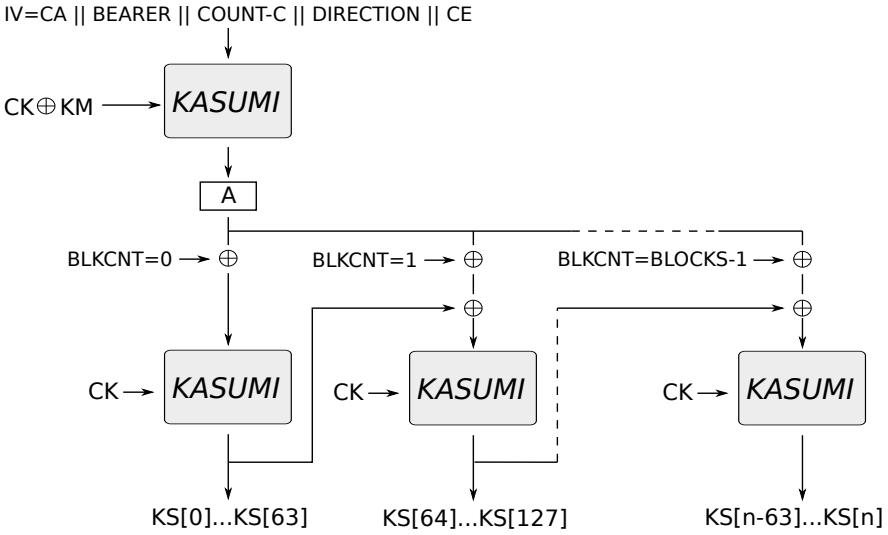


Figure 2.9: UEA1 key-stream generator

To implement the UEA1, it is necessary to transform the KASUMI block cipher into a key-stream generating stream cipher. UEA1 uses a non-standard way to do this which employs *pre-whitening* as well as the cipher block counter (CNT) and output feedback (OFB) chaining modes. As depicted in Fig. 2.9, the KASUMI cipher is initialized with an initialization vector (IV) consisting of the concatenated parameters CA, BEARER, COUNT-C, DIRECTION and CE. Then the $\oplus$ operation used to combine the Cipher Key (CK) with a constant Key Modifier (KM) and the KASUMI algorithm is executed to compute the value $A$. This process is known as *pre-whitening*. In the next stage the value $A$ is combined with the increasing publicly known block counter value (BLKCNT) and KASUMI encryption is performed. The technique of encrypting a counter value with a secret key to generate a key-stream is known as *counter mode (CNT)*. The last step involves feeding back the output of one round as the input of the next round (which is known as *output feedback mode (OFB)*. The final result is the key-stream KS available in chunks of 64-bit each.
In contrast to UEA1, UEA2 utilizes a stream cipher known as *SNOW 3G*. It consists of a Linear Feedback Shift Register (LFSR) combined with a Finite State Machine (FSM) [49]. The FSM consists of three registers and two substitution boxes (S-BOXes). Unlike A5/2, UEA2 is not an

intentionally weakened ciphering algorithm for export use. Instead it was specified by ETSI as *backup* algorithm that can be used in case UEA1 is broken [49]. An overview of the algorithm is given by Debraize and Corbella [49] while the full specification of UEA2 and SNOW 3G is available through the GSM Association (GSMA) [50].

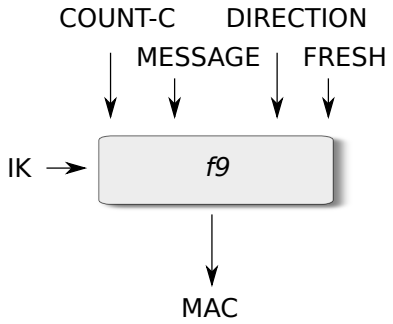


Figure 2.10: $f9$ integrity protection function

Besides encryption, UMTS messages are also integrity protected with the integrity protection key $IK$ and the function $f9$ visible in Fig. 2.10. To compute the Message Authentication Code MAC for a message, the function $f9$ requires the mentioned COUNT-C and DIRECTION parameters for initialization. Furthermore, it requires the key $IK$, the message (MESSAGE) itself and a fresh random value FRESH, which is generated by the network. As only the UE and the genuine network know the secret key $IK$, the function can be used on both sides to prove that a received message is authentic. Like the UMTS Encryption Algorithms (UEA), the UMTS Integrity Algorithm (UIA) utilizes a *kernel* algorithm: UIA1 uses the KASUMI kernel algorithm in CBC-MAC (Cipher Block Chaining MAC) mode while UIA2 uses SNOW 3G.
The UIA1 CBC-MAC construction is visible in Fig. 2.11. To compute the UIA1 Message Authentication Code MAC, the $f9$ input fields COUNT-C, FRESH, DIRECTION and the message itself (MESSAGE) are concatenated and divided into the 64-bit blocks $PS_{[0\ ..\ BLOCKS-1]}$. If the concatenated fields are not divisible into 64-bit blocks an according number of padding bits is added. Each Block is encrypted with the KASUMI algorithm and the secret key $IK$. The encrypted result of each block is then combined with the input of the next block by means of the $\oplus$ operation. Finally, the output of all KASUMI encryptions is combined and encrypted once again with the key $IK \oplus KM$, where KM is a constant key modifier. A more detailed description of the UIA algorithms can be found in [41], [46] and in the corresponding standard documents [48, 50].

### Security

In this section we analyze the security of the UMTS/3G network with respect to Alice and Bob who wish to communicate securely (i.e. the properties *confidentiality*, *integrity* and *authenticity* should hold). Security issues of the UMTS/3G network itself, that do not affect the given set of properties in the communication between Alice and Bob, are not relevant to us.
In comparison to GSM, UMTS has undergone a significant security improvement. In addition

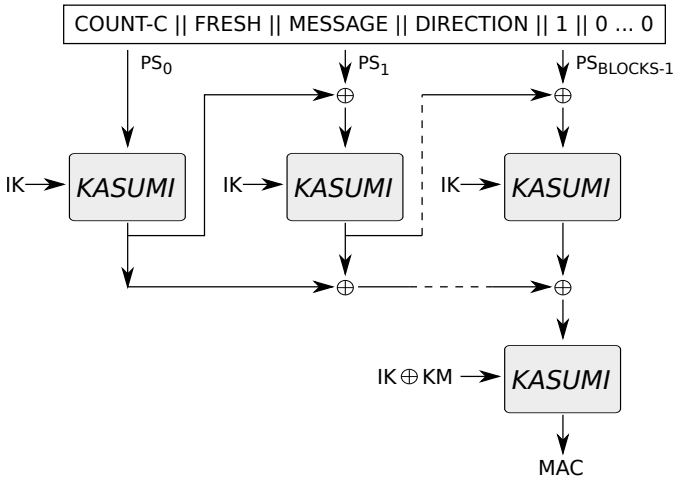


Figure 2.11: UIA1 CBC-MAC construction

to the GSM security features that have turned out to work well, *mutual*-authentication, integrity protection and encryption with publicly scrutinized algorithms were introduced. Nevertheless, with regard to the security properties that Alice and Bob would like to have in their call, it is still not possible for them to have a secure conversation. The reason for this is that most of the UMTS security features hold for the network, but not for the communicating parties.

*Confidentiality* between Alice and Bob can not be achieved, as the encryption is not end-to-end. Instead, speech and signaling data is only encrypted on the Over-The-Air (OTA) interface between the User Equipment (UE) and the Radio Network Controller (RNC). Everywhere else in the network the content of their conversation passes in clear. Likewise, neither *integrity* nor *authenticity* can be guaranteed between Alice and Bob as these properties also only hold for the network, but not for the conversational partners. In fact the integrity protection between the UE and the RNC only applies to signaling-, but not to user-data [51].

Consequently an inside attacker (like a technician) can violate the confidentiality, the integrity and the authenticity of calls between Alice and Bob without notice [28]. Moreover, the stipulated lawful interception (LI) capabilities of the network are specifically tailored to break the confidentiality of targeted phone calls [28], regardless whether used lawfully or not.

However, similar to the section on GSM security (2.1), we argue that mainly due to the easier access methods, the attack potential on wireless network components is broader than it is for internal components. At a first glance successful attacks on the wireless part of the network seem to be unlikely as this is where most of the described security improvements took place. A closer look reveals that there are at least some security implications.

**In a recent publication** by Dunkelman et al. [6] the class of the so called *sandwich attacks* on the KASUMI cipher was introduced. It discredits 3GPP's assurances that the changes they made to MISTY in order to get the "optimized" KASUMI cipher do not have any implications on the cryptographic security. Basically in the design process the ETSI Security Algorithms Group of Experts (SAGE) simplified the key schedule and some components of MISTY to make it faster and more hardware-friendly. They called this new variant KASUMI and stated that

"removing all the FI functions in the key scheduling part makes the hardware smaller and/or reduces the key set-up time. We expect that related key attacks do not work for this structure" [6]. In contrast the new attack technique developed by Dunkelman et al. [6, 52] clearly proves that related key attacks are not only possible, but also practically feasible. Their attack requires $4$ related keys, $2^{30}$ bytes of memory while the data complexity is $2^{26}$, and the time complexity is equivalent to $2^{32}$ KASUMI encryptions. Due to the low complexities they could verify their results experimentally. Using an unoptimized KASUMI implementation on a modest PC they were able to recover "96 key bits in a few minutes, and the complete 128 bit key in less than two hours." [6]. In comparison careful analysis indicates that the original MISTY algorithm is not prone to the attack. To the best of our knowledge the best known attack on MISTY still has an impractically high time complexity of $2^{123}$ [52]. While this shows that the changes made by 3GPP resulted in a much weaker cryptosystem, it has currently no implications on the practical security of A5/3 or UMTS/3G as KASUMI is used in a very specific way there [6, 52].

**With the ongoing transition** from GSM to 3G/UMTS networks, today's 3G communication equipment mostly supports both systems. In areas with no UMTS coverage, it is possible to automatically fall back on GSM. Seamless switching between UMTS and GSM networks is even practicable during a call in so called intersystem UMTS(3G)/GSM(2G) handovers. Meyer and Wetzel [53, 54] have shown that there are security weaknesses in the interface of these two technologies. While the older GSM System is prone to malicious attacks and only supports subscriber authentication, the newer UMTS system provides mutual authentication and integrity protection. The applied UMTS authentication and key agreement (AKA) protocol has been specifically designed to be secure again man-in-the-middle attacks [53, 54]. But if UMTS equipment connects to a GSM-only base station (BSS) there is no support for UMTS security features like integrity protection which allows an attacker to impersonate a GSM BSS to an UMTS subscriber [53, 54, 55]. As a result a man-in-the-middle attack is feasible and the victim's MS/UE can be fooled into using either no encryption or a weak encryption algorithm [53, 54, 28, 55]. Eventually this will allow an attacker to eavesdrop on the communication. Besides passive attacks are feasible in certain GSM/UMTS handover scenarios that may compromise crucial key material [54]. We would like to stress that as long as hybrid GSM/UMTS equipment is in use, an attacker may always exploit the weakest links in the (GSM) security chain with the results having severe consequences on communication confidentiality and integrity. For example, it would be relatively straight forward to jam UMTS coverage in the vicinity of the victim and then present and impersonated GSM BSS as a fallback to the subscriber to mount subsequent attacks.

## 2.3 Conclusion

We showed that the security of GSM can be considered broken today. The wireless connection is only encrypted between the handset and the base station tower (BTS), allowing resourceful or inside attackers to eavesdrop on conversations (e.g. by intercepting directional microwave links or by leveraging the lawful interception capabilities). However, much more severe are the weak encryption algorithms on the wireless channel and the unilateral authentication scheme. We presented the class of passive attacks and the more powerful active attacks. While the

intentionally weak A5/2 encryption should be phased out by now, A5/1 is still being used on GSM handsets, allowing adversaries to mount passive attacks and break the encryption within seconds. In contrast, A5/3 is more secure and, to the best of our knowledge, has not been practically broken so far. Nevertheless, active attacks usually exploit the unilateral authentication scheme by setting up a fake base station. This way encryption can be completely avoided, enabling eavesdropping attacks even if stronger algorithms like A5/3 are employed on the GSM network. UMTS addressed the shortcomings of GSM which led to a more secure system. However, to permit the seamless transition from GSM to UMTS, UMTS includes GSM/UMTS handovers and todays handsets usually support both systems. As a result, UMTS suffers from GSM legacies and adversaries can still mount active attacks to eavesdrop on conversations. Like GSM, UMTS offers no protection against eavesdropping by inside attackers or by use of the lawful interception capabilities.

Persons, who wish to have a secure conversation, can not rely on those networks. They need strong end-to-end encryption that sets up a secure channel over insecure networks like GSM or UMTS. They should be able to put trust into the security of the system, which is only possible by specifying the inherent information security properties and relying solely on established cryptographic primitives, that are practically proven, widely-used and known to be secure. After all, GSM and UMTS are still marketed to be secure and there are dubious “secure phone” products like the Secure Phone Miser[11], that do not offer trustworthy security. It needs to be feasible to make a clear distinction between solutions that offer real security and those that are merely marketed as being secure. Ultimately, real security is only possible with a completely open design.

CHAPTER 3

# Established cryptographic principles and protocols

Whenever information such as the content of a conversation or a letter has to be kept secure, usually a set of certain information security objectives applies. In the previous Chapter 2 we made the assumption that a communication between has to meet the following information security properties:

- Confidentiality
- Integrity
- Authenticity (of entities and messages)

Essentially, these three properties form the core principles of information security and are also known as the "CIA triad". To build a robust communication system that fully meets the requirements of Alice and Bob, a number of additional security objectives and key concepts are relevant [56]:

**Availability**
: Whenever Alice and Bob wish to use the communication system, it should be available for use and working correctly.

**Non-repudiation**
: It should not be possible to deny any previous communication acts. In fact both Alice and Bob should be able to prove to a third party that the information came from a specific entity.

**Signature**
: It should be possible to bind information to an entity (e.g. to Alice or Bob).

**Receipt**
It should be possible to acknowledge that information has been received.

For now we postpone the decision of which of these security objectives might be desirable for a secure real-time speech communication system to a later chapter. Instead we would like to outline that only a certain degree of digital information security can be achieved by technical means provided through cryptography [56]. At the same time, cryptography is not the ultimate solution to information security. There are information security properties (like availability) that can not be achieved through cryptographic techniques (alone) and other security measures such as physical security or security policies might be required. Instead, the fundamental goal of cryptography is to adequately address the following four areas both in theory and in practice [56]:

1. Confidentiality
2. Data integrity
3. Authenticity (of entities and messages)
4. Non-repudiation

To some degree it is feasible to derive other information security objectives (such as having a signature) from these cryptographic areas. The basic cryptographic tools to achieve these information security goals are the *cryptographic primitives* [56]. They can be divided into *unkeyed primitives* such as arbitrary length hash functions, *symmetric key primitives* such as symmetric key ciphers or keyed hash functions and *public key primitives* like public key ciphers or signatures. In the next sections, we will deal with cryptographic primitives, principles and protocols that are relevant when designing a system for real-time encrypted speech communication over low bandwidth channels.

## 3.1 Cryptographic principles

In this section, we will exemplify standardized, well respected and widely used cryptographic principles that seem especially adequate for the application in a low-bandwidth ($\leq 9600$ baud) real-time speech communication environment. In addition to being standardized, well respected and widely used, the principles also need to work well and efficiently on embedded hardware and they need to produce a minimum amount of overhead data.
In digital speech communication, speech is typically sampled, compressed and transmitted in packets over a communication medium. At the receiver the speech frames in the packets are decompressed and played back. When doing so, there is a tradeoff between the number of packets that need to be sent in a given period of time (e.g. 1 second) and the latency. The smaller the number of packets, the bigger the packet payload needs to be. But this comes at the price that it also takes more time until enough speech samples are available to fill up the packet. At the receiver this “waiting time” will be noticeable in the form of additional communication system latency. On the other side each packet needs to contain additional data like a packet header which usually requires more space than a bunch of speech samples. Hence in the extreme case,

minimizing communication latency by only sending a single speech sample per packet would dramatically increase the required communication channel bandwidth.
When designing a secure communication system, usage of cryptographic primitives will coercively introduce additional overhead data and consequently additional latency. Likewise, the low bandwidth of the system will render traditional cryptographic principles like the Diffie-Hellman-Merkle (DH) key exchange, the Digital Signature Algorithm (DSA) or RSA practically unusable. To give an example, the ECRYPT II Yearly Report on Algorithms and Key Lengths [57] recommends the use of $256$-bit symmetric keys to offer sufficient protection for the "foreseeable future" and against quantum computers. But if we plan to use traditional asymmetric cryptographic principles like DH or RSA to achieve $256$-bit security, the asymmetric keys would need to have a size of at least $15424$ bits [57] ($= 1928$ bytes). Assuming that the highest possible speed of our communication channel is limited to $9600$ baud, merely transferring a single key would take more than 2 seconds. By taking into account that during encryption setup usually a number of keys need to be exchanged, the required amount of time would rise even more and the system would quickly become practically unusable.

### Elliptic Curve Cryptography (ECC)

By using Elliptic Curve Cryptography (ECC) the key sizes can be significantly reduced without reducing security [57]. Therefore, especially in a low bandwidth environment the use of Elliptic Curve Diffie-Hellman (ECDH) key agreement and Elliptic Curve Digital Signature Algorithm (ECDSA) is favorable over traditional DH and DSA.
Table 3.1 gives an overview on equivalent symmetric, asymmetric and ECC key lengths in accordance with ECRYPT II (2010) [57].

| symmetric | asymmetric | ECC |
|---|---|---|
| 80 | 1248 | 160 |
| 96 | 1776 | 192 |
| 112 | 2432 | 224 |
| 128 | 3248 | 256 |
| 256 | 15424 | 512 |

Table 3.1: equivalent key size comparison in bits

To leverage elliptic curve cryptography, initially Alice and Bob need to agree upon the elliptic curve *domain parameters* which are the parameters that define the elliptic curve. Generally, an elliptic curve can be defined through a *Weierstrass equation* [58]

$$y^2 = x^3 + ax + b$$

whereas the coefficients $a$ and $b$ are real numbers that need to fulfill the constraint

$$4a^3 + 27b^2 \neq 0$$

to avoid singularities (i.e. the curve has no self-intersections, isolated or angular points).
By choosing $a$ and $b$ accordingly, an arbitrary number of different elliptic curves can be defined. To be able to use elliptic curves for cryptography, the elliptic curve and its mathematical operations (such as point addition or multiplication) need to be defined in finite *Galois Fields* (GF). For Elliptic Curve Cryptography (ECC) either prime $GF_p$ or binary $GF_2^m$ fields can be used [58, 59] as long as the fields are chosen with a finitely large enough number of points for cryptographic operation. As elliptic curve point multiplication dominates execution time [60], the overall performance depends on what type of curve is used. Generally speaking elliptic curve multiplication in binary $GF_2^m$ fields is better suited for hardware implementation whereas for prime $GF_p$ fields a software implementation might be more appropriate [59].
Elliptic curves on $GF_p$ are defined as

$$y^2 \; mod \; p = x^3 + ax + b \; mod \; p$$

where

$$4a^3 + 27b^2 \; mod \; p \neq 0$$

As a result all curve elements in $GF_p$ are integers in the range $[0; p-1]$. In contrast, elliptic curves on $GF_2^m$ are defined as

$$y^2 + xy = x^3 + ax^2 + b$$

where

$$b \neq 0$$

which results in all curve elements in $GF_2^m$ being integers with a length of at most $m$ bits.
In addition to the curve parameters that we already defined, for ECC a *generator point* $(x_G, y_G)$, the order of the curve $n$ and the cofactor $h$ are required as well. The generator point $(x_G, y_G)$ is a specific chosen point on the elliptic curve that will be used for cryptographic operations. The order $n$ of the curve is required for point multiplication, as the scalar is chosen to be in the range $[0; n-1]$.
In summary, this leads to the following domain parameters for ECC in $GF_p$:

**(p, a, b, G, n, h)**.

Instead of the prime number $p$, in $GF_2^m$ the parameters $m$ and $f(x)$, an irreducible polynomial of degree $m$, are necessary. Therefore, the ECC domain parameters in $GF_2^m$ are:

**(m, f(x), a, b, G, n, h)**

Domain parameters are usually not generated by the users themselves as the process involves counting the number of points on each curve which is time-consuming [61, 60, 62]. Instead a number of curves was defined by standardization organizations such as NIST [63] or Certicom's Standards for Efficient Cryptography Group (SEGC) [62]. The advantage of these curves is that

they are publicly scrutinized and special care has been taken to avoid weak classes of curves that are vulnerable to a number of attacks [56].
The security of Elliptic Curve Cryptography (ECC) mainly depends on the Elliptic Curve Discrete Logarithm Problem (ECDLP) [61, 58].
Assume that $P$ and $Q$ are two points on an elliptic curve defined in a finite Galois field $GF$ of large order $n$, such that

$$dG = Q$$

where $d$ is a sufficiently large scalar value. Then it is easy to compute $Q$ out of $d$ and $G$, but it is computationally infeasible to determine $d$ if only $G$ and $Q$ are given [61, 58]. This is also the reason why during cryptographic ECC operations mainly elliptic curve point multiplication is used. By directly applying the Elliptic Curve Discrete Logarithm Problem (ECDLP), an asymmetric key pair can be obtained (see Section 3.1).
In general Elliptic Curve Cryptography (ECC) as well as the cryptographic protocols ECDH and ECDSA are standardized by well respected institutions such as NIST (FIPS 186-2), ANSI (ANSI X9.62, ANSI X9.63), ISO/IEC (15946) and IEEE (IEEE P1363) [61, 59]. In addition they are also included in NSA's Suite B Cryptography [64] for the protection of classified information up to the highest security clearance level *top secret*.

### Elliptic Curve Diffie-Hellman (ECDH)

In contrast to the traditional DH exchange, ECDH may be implemented using two different primitives [57]. The elliptic curve Diffie-Hellman primitive is comparable to the traditional DH exchange, whereas the elliptic curve cofactor DH primitive provides additional resistance to attacks like small subgroup attacks by including the cofactor into the shared secret calculation [57].
The basic idea of both primitives is the same: Two parties $U$ and $V$ can compute a shared secret value. That is $U$ supplies her secretly kept private key and $V$'s public key as input to the primitive in order to compute the shared secret. Vice versa, $V$ executes the primitive with her private key and $U$'s public key to compute the same shared secret.
To obtain a key pair (i.e. a private and a public key), $U$ and $V$ first need to agree on a common set of domain parameters. When doing so, both parties need to verify that these parameters are in fact valid by using a set of validation procedures [60]. Once validated, a key pair can be generated in the following way:

**Algorithm 3.1** ECC key pair generation

1: select a random integer $d$ in the interval $[1; n-1]$, where $n$ is the order of the curve
2: calculate $Q = dG$, where $G$ is the *generator point*
3: Output $(d, Q)$

The outcome is a key pair $(d, Q)$ with $d$ being the private and $Q$ being the public key. Assuming that $U$ and $V$ have a common set of domain parameters, $(d_u, Q_u)$ is the key pair of $U$ and $(d_v, Q_v)$ is the key pair of $V$, $U$ and $V$ can utilize the ECDH primitive [60, 58]:

**Algorithm 3.2** ECDH shared secret calculation

1: $U$ and $V$ exchange their public keys ($Q_u$ and $Q_v$ respectively)
2: $U$ calculates an elliptic curve point $P = (x_P, y_P) = d_u Q_v$
3: if $P = 0$ output "invalid" and stop
4: $V$ calculates an elliptic curve point $R = (x_R, y_R) = d_v Q_u$
5: if $R = 0$ output "invalid" and stop
6: since $d_u Q_v = d_u Q_v G = d_v Q_u G$, also $P = R$ (i.e. $x_P = x_R$ and $y_P = y_R$)
7: $U$ and $V$ have a shared secret $x_P = x_R$

Note that if instead of regular ECDH, the elliptic curve cofactor DH primitive is used, in step 2 and 4 the cofactor $h$ has to be included in the point calculation (i.e. $U$ would compute $P = (x_P, y_P) = h d_u Q_v$).

### Elliptic Curve Digital Signature Algorithm (ECDSA)

Similar to DSA, the Elliptic Curve Digital Signature Algorithm (ECDSA) is a digital signature scheme that offers a *sign* and *verify* operation usable to provide the information security objectives data and entity authentication, integrity protection and non-repudiation. We can divide ECDSA into a *signature generation* and a *signature verification* algorithm.
Like in the previous section, we assume that there is an individual $U$ already having a valid ECC key pair $(d_U, Q_U)$. To sign a message $m$ with ECDSA, $U$ can use her private key $d_U$ [61, 58]:

**Algorithm 3.3** ECDSA signature generation

1: select a random integer k in the interval $[1; n-1]$, where $n$ is the order of the curve
2: compute $kG(x_1, y_1)$, where $G$ is the *generator point*
3: compute $r = x_1 \; mod \; n$
4: if $r = 0$ goto step 1
5: compute $e = H(m)$, where H is a cryptographic hash function (e.g. SHA-1)
6: compute $s = k^{-1}(e + d_U r) \; mod \; n$, if $s = 0$ goto step 1
7: Output signature $(r, s)$

A potential receiver $V$ of message $m$ can verify $U$'s signature $(r, s)$ if she knows $U$'s ECC domain parameters and the public key $Q_U$:

**Algorithm 3.4** ECDSA signature verification

1: for the signature $(r, s)$, verify that $r$ and $s$ are in the interval $[1; n-1]$
2: compute $e = H(m)$, where H is a cryptographic hash function (e.g. SHA-1)
3: compute $w = s^{-1}\ mod\ n$
4: compute $u_1 = ew\ mod\ n$
5: compute $u_2 = rw\ mod\ n$
6: compute $X = u_1G + u_2Q$, where $G$ is the *generator point*
7: if $X = \infty$, output "invalid" and stop
8: compute $v = x_1\ mod\ n$
9: if $v = r$, the signature is correct, otherwise output "invalid" and stop

After successful verification, $V$ knows that the message $m$ really came from $U$ (*entity authentication*) as only $U$ knows the secret key $d_u$ that is required to compute the signature $(r, s)$. Moreover, $V$ knows that the message itself is authentic and it has not been tampered with (*message authentication* and *integrity protection*) as the signature was calculated over the cryptographic hash value $e$ of the message $m$. (We assume at this point that the cryptographic hash function $H$ is secure.) Finally, $V$ can use the signature to prove to a third party, that the message was indeed sent by $U$ and the message content has not been tampered with (*non-repudiation*) which also means that $U$ can not disclaim to have sent the message with the given content.

### Hash-based Message Authentication Code (HMAC)

In general, cryptographic hash functions can be distinguished into *unkeyed* and *keyed* hash functions [56]. Unkeyed hash functions are mainly used for modification detection. For instance, a sender could send a file to a receiver and attach the calculated hash value to it. The receiver calculates the hash value as well and compares it with the received one. If they match, the file was correctly received, otherwise the file was modified during transmission.
Likewise, unkeyed cryptographic hash functions are essential for digital signatures. In ECDSA for example, an unkeyed cryptographic hash value is computed over a message $m$ which is then *signed* with the private key. This way a digital signature can be applied to the actual *content* of the message. The signature provides *data* and *entity authentication* as well as *integrity protection* and *non-repudiation*.
With keyed hash functions such as the Hash-based Message Authentication Code (HMAC) a very similar set of information security objectives can be achieved. As a secret key is required to compute the hash value of a message, only entities that have the correct key can compute the correct hash value. Suppose that a group of individuals have securely exchanged a Message Authentication (MAC) key. Then, for any message they exchange, they can compute the keyed hash value and, as a result, verify that the message was sent by an authentic member of the group, as only someone in the group is in possession of the shared secret key. This way *data* and *entity authentication* and *integrity protection* can be provided, but no *non-repudiation* as anyone in the group could have generated the correct hash value. This, and the fact that Message Authentication Codes (MACs) can be used with a single key by a *group* of individuals, is the big difference in comparison to public key digital signature schemes like ECDSA.

The Hash-based Message Authentication Code (HMAC) is standardized by NIST in FIPS 198 [65], ISO/IEC (ISO/IEC 9797-2) [57] and by the IETF in RFC 2104 [66]. It is also widely used in protocols such as SSL, TLS or IPsec [57].
The $HMAC(k, m)$ with key $k$ and message $m$ can be computed by utilizing a core cryptographic hash function $H$ (such as SHA-1 or SHA-256) with

$$HMAC(K, m) = H((K_0 \oplus opad) \mid\mid H(K_0 \oplus ipad) \mid\mid m))$$

where $||$ denotes concatenation. As the size of $K$ needs to equal to the block length $B$ of the core hash function $H$, $K_0$ can be obtained either by truncating $K$ (if the length of $K > B$) or by appending zero bytes (if the length of $K < B$) [65, 66].
The values $opad$ and $ipad$ are constants with length $B$ [66]:

| | |
|---|---|
| $ipad$ | the byte 0x36 repeated B times |
| $opad$ | the byte 0x5C repeated B times |

Table 3.2: HMAC $opad$ and $ipad$ padding constants

If $L$ is the length of core hash function $H$'s output, the size of the key $K$ "shall be equal to or greater than $\frac{L}{2}$" [65]. Contrary to that suggestion, RFC 2104 recommends, that the size of the key $K$ should be at least $L$ [66]. Keys having a greater length do not significantly increase the security [65]. Due to the construction of the HMAC, the security mainly depends on the size of the secret key and the security of the core hash function $H$ [66].
Besides, the FIPS 198 standard [65] defines HMAC *truncation*, which is a well known practice that can be applied whenever there are size or speed constraints. For example, on a low bandwidth channel, it might not be feasible to append the full 32 bytes long HMAC-SHA256 tag to each message. Instead, with truncation, only the $t$ leftmost bytes of the HMAC output are used for each message. As this also reduces the security of the HMAC, the FIPS 198 standard recommends that "$t$ shall be at least $\frac{L}{2}$ bytes [...] unless an application or protocol makes numerous trials impractical" [65]. Thus the minimum length of $t$ is closely related to the application: While for a low bandwidth channel precluding too many trials a 32 bit HMAC tag can be sufficient [65], other applications might require longer HMAC tags.

### Advanced Encryption Standard (AES) in Counter Mode (CTR)

The established Advanced Encryption Standard (AES) is a very widespread and well known publicly scrutinized block cipher standardized in NIST FIPS PUB 197 [67], in ISO/IEC 18033-3 and it is also part of NSA's Suite B Cryptography [57, 64] approved for the protection of information up to the highest *top secret* security level. Instead of going too much into the details about how AES works (interested readers are referred to the Federal Information Processing Standards Publication 197 [67]), we would like to deal with the challenges of using AES in a low bandwidth constrained communication environment.
As any block cipher, AES only allows to encrypt and decrypt *blocks* of data with a predefined length. Regardless of the key size (i.e. 128, 192 or 256 bits), AES always operates on blocks

of 128 bit (16 byte) length. For that matter if Alice plans to send a secret 1 byte long message to Bob, she would have to add 15 padding bytes to the message before being able to encrypted it with AES. This may not be an issue for applications with large amounts of data (e.g. disk encryption), but it basically renders AES (or any other block cipher) unusable for low bandwidth speech communication systems that usually only send small chunks of data. In that case much of the existing bandwidth would be consumed by those padding bytes.
Instead, for this application, the use of a *stream cipher* seems to be more adequate as the length of data does not increase through encryption. A stream cipher basically generates a continuous key stream that is used to encrypt plaintext bytes by applying the $\oplus$ operation. Unfortunately, some of the most widely used stream ciphers (such as A5 or RC4 in the case of WEP) turned out to be insecure.
An elegant way to leverage the benefits of a stream cipher together with the security and robustness of a well established block cipher like AES is the use of *Counter Mode (CTR)*. Counter Mode (CTR) effectively transforms any block cipher into a stream cipher by encrypting a steadily increasing *counter value* instead of the plaintext data itself [57, 68, 69]. The output of the block cipher is used as a key-stream, whereas the $\oplus$ operation is applied to produce the ciphertext from the plaintext. In the case of AES, the counter usually has at least the size of the cipher's block size (i.e. 128 bit) [69] which guarantees that it won't repeat for a long enough time. In addition to the key, also an Initialization Vector (IV) is mandatory [57, 68, 69] that needs to be unpredictable for the attacker [57]. It is of highest importance to eschew key-stream reuse by not using the same combination of IV and key more than once [57, 68].
Today, Counter Mode (CTR) is a NIST (SP-800-38A) and ISO/IEC (10116:2006) standardized operation mode [57], that is widely used in protocols such as Secure Shell (SSH) [69] or IPSec [68]. It can be proven that if the underlying block cipher is secure, also CTR mode is secure [57]. This way using the Advanced Encryption Standard (AES) in Counter Mode (CTR) allows us to rely exclusively on well-established, standardized and secure encryption techniques in the field of low-bandwidth encrypted speech communication.

## 3.2 Key exchange in existing cryptographic protocols

If two individuals (Alice and Bob), not having shared any key material, wish to establish a secure communication with each other, they need to exchange keys at some point. While existing key exchange mechanisms like Diffie-Hellman (DH) are readily available, they can often be compromised by an active attacker. In case of Diffie-Hellman (DH), the key agreement scheme suffers from a well known Man in the Middle (MitM) attack in which Alice and Bob believe that they establish a key with each other, but in reality they establish (differing) keys with a third party (the attacker Eve) in the middle. For that reason, it does not suffice to do a simple key exchange, but Alice and Bob also need to be sure that the key exchange is *authentic*.
Under the assumption that Alice and Bob have no key material from each other and they do not trust anyone else (like a trusted third party), in the following part we will have a look at how existing protocols tackle the key exchange authentication problem.

### Secure Shell (SSH)

The Secure Shell (SSH) protocol is one of the prevailing protocols for remote login on UNIX machines. If a key exchange is executed in the SSH protocol, it is digitally signed with the private host key of the server to provide authentication [70]. Hence the client needs the server's public host key to verify that it is really talking to the correct server. Yet if the client does not already possess the public key and no trusted certification authority (CA) is available, it is suggested that out-of-band key verification is used instead [71].
In SSH this is usually done by calculating a hexadecimal fingerprint derived from the SHA-1 hash of the host's public key. Thereupon these fingerprints can be verified "by using telephone or other external communication channels" [71]. If the fingerprint is not verified or the external communication channel is under control of an attacker, the initial key exchange might be susceptible to a Man in the middle (MitM) attack. Once the client knows the server's public key, it is used to verify the authenticity in subsequent connects [71], a concept also known as key continuity management [72]. As a result, in case of a successful MitM attack on the initial key exchange, the adversary would have to mount the attack on each successive key exchange as well, to avoid being detected.

### ZRTP

ZRTP is a protocol originally developed by Phil Zimmerman that can be used to negotiate keys between two parties in a Voice over Internet Protocol (VoIP) setup based on the Real-time Transport Protocol (RTP) [13], whereas the key exchange is based on Diffie-Hellman (DH).
The ZRTP protocol introduces a "Short Authentication String (SAS)" [13] to authenticate the Diffie-Hellman key exchange. Essentially, the SAS is a truncated keyed cryptographic hash value (HMAC) of the exchanged Diffie-Hellman parameters [13, 73]. During the first step towards key material negotiation, a hash commitment value is sent that obliges the initiator not to change his Diffie-Hellman key pair [73]. That is "the responder chooses his keys before knowing the initiator keys and the initiator chooses his key before sending the hash commitment, that binds him to that choice" [73]. Accordingly the agents can not deterministically influence the SAS [73] and the attacker has only one guess to generate the correct SAS value [13]. It is argued that for this reason, despite the SAS value being rather short (e.g. 16-bit), the probability for the attacker not being detected is very low (i.e. one out of $65536$ for a 16-bit SAS value) [13].
Under the assumption that the communicating parties can distinguish voices, the SAS can be verbally compared by two users by "using their human voices, human ears, and human judgement" [13]. If the SAS does not match, it indicates the presence of a MitM attack. Similar to SSH, after the initial key exchange, a key continuity scheme is applied and SAS authentication is no longer needed. Although the security of the ZRTP protocol has been formally proven (with the assumption that SAS is secure) [73], theoretical attacks on the SAS authentication exist mainly if the attacker has voice impersonation capabilities, Alice and Bob do not know the voice of each other or they have problems recognizing the voices (e.g. due to background noise or bad audio quality on the medium) [74, 75]. Yet in practice the authentication mechanism is considered secure [76] and actively used in a number of both open-source and commercial secure phones [13, 76, 12].

### Off The Record (OTR) protocol

The Off The Record (OTR) protocol is a protocol that allows instant messaging users to have private conversations with each other. It is supported either natively or through plugins by widely used instant messaging application such as Adium, Miranda IM, Pidgin, Trillian and others.
In the first version of the OTR protocol [77] during the initial Diffie-Hellman key exchange a fingerprint of the other party's public key is displayed. Similar to SSH, the user is supposed to verify the fingerprint out-of-band so that a successful MitM attack can be prevented. The protocol also involves a key continuity scheme to allow authentication of subsequent connections with the help of the verified public keys.
It has to be noted that OTR only uses signatures to authenticate the shared Diffie-Hellman secret while subsequent encrypted content is authenticated using Message Authentication Codes (MACs). In the second version of the protocol [1, 15], the initial Diffie-Hellman key exchange was dropped in favor of a modified SIGMA [78] variant, which resolves a previously discovered identity mis-binding attack on the initial DH key exchange and hides the public keys of the participants from passive adversaries. Furthermore, the authentication procedure was extended with an adapted version of the Socialist Millionaire's Protocol (SMP) that allows two parties to verify that they both know a preshared secret without actually revealing the secret [1]. The SMP protocol leverages a number of zero-knowledge proofs to demonstrate the correctness of values that are exchanged throughout the protocol. It is assumed that most parties already share some kind of secret information that is not available to an attacker (e.g. the answer to the question "Where did we first meet ?").
Together with the fingerprint of the exchanged public key this pre-shared secret information can be used with SMP to verify the authenticity of the fingerprint and of each other.

## 3.3 Conclusion

We showed that there are established cryptographic principles and protocols that are standardized, widely used and practically proven to be secure. Algorithms based on Elliptic Curve Cryptography (ECC) are especially suitable for low-bandwidth application, as the required key sizes are significantly shorter than for their asymmetric counterparts. They reduce the amount of data that needs to be transferred over low-bandwidth channels and consequently, the duration of those data transfers can be minimized. For integrity protection and data encryption we described the HMAC function and AES in Counter Mode. With the help of HMAC, message authentication codes are created, whereas through standardized truncation procedures we can choose the length of the hash in accordance with our system and security requirements. The standardized Counter Mode effectively transforms AES into a stream cipher, which is optimal for encrypted speech communication, as arbitrary length blocks can be encrypted without the need of padding bytes. However, designing a secure cryptographic protocol is not a straight forward task, even if only secure principles are used throughout the protocol. It usually involves numerous reviews and improvements by the cryptographic community until a cryptographic protocol is secure. Therefore we looked at the key exchange of publicly scrutinized cryptographic protocols to form the basis for our own key exchange, protocol design and security features.

CHAPTER 4

# Protocol design and security features

## 4.1 Desired properties of the communication system

So far we discussed which information security objectives can be achieved through cryptographic techniques and protocols (see Chapter 3). Yet we did not decide upon which objectives our encrypted speech communication system must fulfill. Objectives like *availability* might be essential as well, but as they can not be achieved through cryptography or communication protocols, it is out of the scope of this thesis.
To make the decisions, we first need a conversational model. Imagine that Alice and Bob know each other and they want to have a private talk. To do so, they might meet in a public park and talk to each other face to face. If no one is listening nearby (which they would possibly see) and neither Alice nor Bob reveal the content of the private talk to anyone else, Alice and Bob can make a few assumptions.
They can assume that

1. they are who they pretend to be (i.e. they recognize each other through their face and voice)
2. no one else can hear what they are talking
3. no one else knows what they talked about
4. no one can make any proofs on what was said to anyone

When designing a secure digital communication system, the scenario is a lot different of course. Instead of meeting personally, Alice and Bob have a phone conversation. In this case the above assumptions must be weakened or even discarded. While Alice might recognize the voice of Bob, she could still be fooled by someone else pretending to be Bob. Neither Alice nor Bob really know whether someone tapped the phone line and is listening to their conversation. Consequently, they can not be sure that they are the only persons who know what was talked about. Furthermore,

they don't know if a recording of the tapped phone line exists, that could prove to others what they talked about.
Hence Alice and Bob would like to have a secure communication system that allows them to safely make the same assumptions they made for the private conversation in the park, while the system utilizes an insecure medium like the telephone line or a cellphone connection.
Obviously, such a system needs to provide *confidentiality* through encryption, *integrity protection* and *authenticity* of data and entities. This way Alice and Bob know that no one listening on the phone line can hear what they are talking about (as the conversation is encrypted) and for the same reason they can assume that no one else knows what they talked about. Likewise, through entity and data authentication they know that they are really talking to each other and due to the integrity protection they can also be sure that the conversation has not been tampered with. However, there still needs to be a way to make it impossible for anyone to make any proofs on the conversation. This leads us to a few more information security requirements that the resulting system must meet:

**Repudiation**

Alice and Bob do *not* want that anyone (including themselves) can make any proofs on the conversation to anyone else. This is contrary to what traditional digital signature schemes like DSA provide, as the verification of a signature always allows to prove to anyone in possession of the public key, that the message came indeed from the signer (who is the sole owner of the corresponding private key) [77, 1].

By purposely granting repudiation, also a certain degree of *plausible deniability* is achieved as, just like for a conversation in the park, Alice and Bob can deny what they have said to each other (for example in case it would put pressure on them).

**Perfect Forward Secrecy (PFS)**

By utilizing message encryption, confidentiality between Alice and Bob is ensured, so that they are the only persons who can read the messages from each other. Yet at a later point in time it might be possible that secret key material is leaked if for example one of the secure communication devices is stolen. In that case it should still be impossible that an attacker can decrypt past messages.

If Perfect Forward Secrecy (PFS) is implemented properly by using short term encryption keys that are discarded after use, not even Alice and Bob can decrypt messages from past conversations [56]. This way they can plausibly deny knowledge of the decryption keys and past conversations are secure even if Alice and Bob are forced to reveal all of their *secret* key material.

While existing protocols such as ZRTP [13] implement at least some of these properties, to the best of our knowledge none of them are suited to be used over channels with very low bandwidth (i.e. 9600 baud and below). For instance, ZRTP is heavily tied to RTP (Real-time Transport Protocol) [13], which would introduce additional data overhead to our communication system. It merely implements the key agreement protocol and the use of SRTP (Secure Real-time Transport Protocol) [79] is mandatory for encryption. Even if SRTP seems to be suited "especially for

voice traffic using low bit rate voice codecs" [80] and "it can be used in conjunction with header compression" [80], large header fields and unnecessary complexity render it unusable for our application.
For this reason, it is indispensable that we design a protocol, that meets the above mentioned properties and still works over low bandwidth channels. Leveraging only the practically proven cryptographic principles and protocols we discussed earlier (see Chapter 3), we could easily start off with designing our own cryptographic protocol. However, as we have seen in existing protocols (e.g. OTR [1, 81, 82]) or in Chapter 2, designing a secure cryptographic protocol is not a straight forward task. Even the most subtle faults in the design can lead to devastating effects on the overall security of the system.
Instead, for a cryptographic system to become secure, usually a (potentially long) process of public scrutiny and improvements by the cryptographic community is necessary. To design a solid and secure communication system, we need to eschew new designs that have not undergone public scrutiny and we need to stay away from developing tailored cryptographic protocols (whereat with "cryptographic protocol" we mean the formal cryptographic protocol specification and not the network protocol carrying the cryptographic messages). Luckily, as we will see in subsequent sections, it is possible to design a system with the properties defined above by solely building upon secure and established protocols.

## 4.2 Adapting the OTR Authenticated Key Exchange for low bandwidth usage

The authenticated key exchange (AKE) in the Off The Record (OTR) [1, 15] messaging protocol provides some of the necessary information security properties that we defined in Section 4.1. The basic idea of the OTR key exchange is to first perform an *unauthenticated* DH key exchange to set up an encrypted channel and then do mutual authentication *inside* that channel [15].
To perform this task, the OTR AKE makes use of the SIGMA signature-based authenticated DH exchange [78, 1, 82] that was "adopted as the main key-exchange protocol in IKE" [82], the Internet Key Exchange version 1 and 2 [83, 84] that is widely used in IPSec. Also, the protocol has been formally analyzed and proven to be secure by R. Canetti et al. [85]. However, neither the OTR protocol nor its Authenticated Key Exchange (AKE) are directly suitable for application in a low bandwidth environment. The main reason for this is that with respect to the slow communication channel between Alice and Bob, the amount of data required to be transferred during the key exchange is too high. Hence the key exchange and ultimately the call setup of the communication system would take too long to be practicable. In order to use the key exchange in our low bandwidth environment, we need to make technical adaptations. When doing so, extreme care needs to be taken that the formal cryptographic key exchange protocol and its cryptographic properties thereof remain unchanged. That is, under no circumstances our technical adaptations should have any implications on the security of the key exchange.

### Detailed description of the key exchange

In the following we will first have a detailed look at the key exchange and then discuss which and how various information security properties are achieved.

**Protocol 4.1** OTR Authenticated Key Exchange (AKE) [1]

1: Alice randomly selects a 128 bit value $r$, where $r$ is the key for the hash commitment
2: Alice randomly selects a 320 bit value $x$, where $x$ is Alice's secret DH key
3: Alice sends $\text{AES}_r(g^x)$, $\text{SHA256}(g^x)$ to Bob
4: Bob randomly selects a 320 bit value $y$, where $y$ is Bob's secret DH key
5: Bob sends $g^y$ to Alice
6: Alice computes $s = (g^x)^y$, where $s$ is the shared DH secret
7: Alice sends $r$ to Bob, so that Bob can *open* the hash commitment
8: Bob decrypts $g^x$ using $r$
9: Bob calculates $\text{SHA256}(g^x)$ and verifies that it agrees with $\text{SHA256}(g^x)$ received in step 3
10: Bob computes $s = (g^x)^y$, where $s$ is the shared DH secret
11: Alice derives MAC keys $a_1, a_2, b_1, b_2$ and AES keys $a_3, b_3$ from $s$
12: Alice selects a serial number $keyid_A$ associated with $g^x$
13: Alice computes $M_A = MAC_{a1}(g^x, g^y, v_A, keyid_A)$, $v_A$ being Alice's public *verify* key
14: Alice computes $X_A = v_a, keyid_A, sign_A(M_A)$, where $sign_A$ is Alice's digital signature
15: Alice sends $\text{AES}_{a_3}(X_a), MAC_{a_2}(\text{AES}_{a_3}(X_a))$ to Bob
16: Bob derives MAC keys $a_1, a_2, b_1, b_2$ and AES keys $a_3, b_3$ from $s$
17: Bob uses $a_2$ to verify $MAC_{a_2}(AES_{a_3}(X_a))$
18: Bob uses $a_3$ to decrypt $\text{AES}_{a_3}(X_a)$ and obtains $X_A = v_A, keyid_A, sign_A(M_A)$
19: Bob computes $M_A = MAC_{a1}(g^x, g^y, v_A, keyid_A)$
20: Bob uses $v_A$ to verify $sign_A(M_A)$
21: Bob selects a serial number $keyid_B$ associated with $g^y$
22: Bob computes $M_B = MAC_{b_1}(g^y, g^x, v_B, keyid_B)$, $v_B$ being Bob's public *verify* key
23: Bob computes $X_B = v_B, keyid_B, sign_B(M_B)$, where $sign_B$ is Bob's digital signature
24: Bob sends $\text{AES}_{b_3}(X_B), MAC_{b_2}(AES_{b_3}(X_B))$ to Alice
25: Alice uses $b_2$ to verify $MAC_{b_2}(AES_{b_3}(X_B))$
26: Alice uses $b_3$ to decrypt $\text{AES}_{b_3}(X_B)$ and obtains $v_B, keyid_B, sign_B(M_B)$
27: Alice computes $M_B = MAC_{b_1}(g^y, g^x, v_B, keyid_B$
28: Alice uses $v_B$ to verify $sign_B(M_B)$

In step 1 of the protocol, Alice chooses an AES encryption key so that she can encrypt her DH public key $g^x$. This is, however, an "engineering requirement" that was necessary in the OTR protocol as "many IM protocols enforce a maximum size on messages" [1].
In steps 2 to 10 Alice and Bob perform a regular (unauthenticated) Diffie Hellman key exchange with the only difference being the hash commitment that ensures that neither party can base their choice of $g^y$ on the other party's chosen $g^x$ value [1]. This allows the OTR secure session ID (i.e. the fingerprint) to be short, while it still makes a successful Man in the Middle (MitM) infeasible [15]. Once step 10 is completed, Alice and Bob have agreed on a shared secret $s$.
In step 11 Alice derives all required MAC and encryption keys from the shared secret key $s$, for

instance by hashing $s$ in various ways. Since all key material is derived from $s$, we refer to that key as *master secret key*.
In the next step 12 Alice selects a serial number $keyid_A$ associated with the current choice of $g^x$. OTR requires these $keyid$'s for re-keying in the protocol.
Step 13 denotes the main part of the SIGMA [78, 82] protocol by calculating the MAC $M_A$ over the DH public keys $g^x$, $g^y$ as well as over Alice's current key-id $keyid_A$ and her long-term public key $v_A$ that can be used to verify her digital signature. At this point we can see that the MAC key $a_1$ is only used once to compute $M_A$, whereas $a_2$ is generally used to authenticate transmitted messages in later steps of the cryptographic protocol. When generating the digital signature $sign_A(M_A)$ in step 14, the use of the MAC key $a_1$ to compute $M_A$ actually proves to Bob that Alice was indeed able to calculate the shared secret $s = (g^x)^y$. According to Raimondo et al. this "prevents the identity mis-binding attack and, at the same time, it provides a deniable exchange by avoiding signing the peer's identity" [82].
In step 14 Alice creates the message $X_A$ comprising her long-term public key $v_A$, her current key-id $keyid_A$ and the digital signature $sign_A(M_A)$ of the previously calculated MAC $M_A$.
Finally, in step 15, she encrypts and sends the message $\text{AES}_{a_3}(X_a)$ together with the corresponding authentication tag $MAC_{a_2}(\text{AES}_{a_3}(X_a))$ to Bob.
Similar to Alice, in step 16 Bob derives the required key material from the master secret key $s$.
In step 17 he verifies that the message from Alice is authentic by checking the authentication tag and, if it was, Bob decrypts it in step 18. This way he obtains Alice's key-id $keyid_A$, her long-term public key $v_A$ as well as the DH public keys, that have been used to compute the shared master secret key $s$.
In steps 19 and 20 Bob uses the information to compute the MAC $M_A$ and verifies its digital signature by means of Alice's long-term public key $v_A$. If everything was correct, in the remaining steps 21 to 28 Bob preforms the SIGMA procedure with Alice.

### Properties of the key exchange

At the end of the protocol, Alice and Bob have a shared secret key $s$, as well as derived key material for message authentication or encryption. "Alice [...] knows Bob's public key $v_B$ , and is convinced that Bob knows the corresponding private key $sign_B$. Bob has a similar assurance about Alice" [1]. In addition, as the transfer of all messages (except for the initial DH key exchange) was encrypted, information such as the long-term public keys or key-IDs is concealed from passive adversaries [1].
With regard to the desired information security properties that we defined in Section 4.1, the OTR Authenticated Key Exchange (AKE) provides key material for encryption ($a_3$, $b_3$), for entity authentication ($v_A$, $v_B$) and for message authentication($a_2$, $b_2$). While it forms the basis for the information security properties *confidentiality* and *(message) authenticity*, it already implements *entity authenticity* as both entities sign the key exchange with their private long-term keys [1, 82].
All key material is derived from the short-term master secret $s$. If an encrypted session is over, the keys are discarded. However, as previous encrypted sessions used short-term key material that is now irrevocably discarded, it is no longer possible to decrypt these sessions even if an attacker gets hold of the encryption devices and the long-term private keys. This gives the key exchange the information security property of *Perfect Forward Secrecy (PFS)* [1, 82].

Apart from that, not even Bob or Alice could decrypt previous communications as they no longer possess the randomly chosen Diffie-Hellman secret keys for past conversations. This grants them a certain degree of *plausible deniability* (i.e. they can plausibly deny to have the ability to decrypt their own past conversations).
Another important property of the key exchange is that is allows *repudiation*. As subsequent messages are authenticated by Message Authentication Codes (MACs), anyone knowing the correct MAC keys (i.e. Alice and Bob) could have created a valid message. Since only the key exchange, but not the subsequent messages are digitally signed [1, 82], neither Alice nor Bob can make any proofs that a message came from a specific party. Once an encrypted conversation session is over, Alice and Bob discard their key material. As this includes the MAC keys that were used to authenticate the messages, Alice and Bob can not even prove that a message came indeed from them.

### Modifications for low bandwidth application

To adapt the key exchange to our needs without causing implications on the security, we applied two methods.
First, we removed all *engineering requirements* from the protocol as they do not affect the security or the cryptographic properties of the key exchange. One such engineering requirement is the encryption of the DH public key in steps $1, 3$ and $8$ (see Protocol 4.1). According to Alexander and Goldberg, the original designers of the OTR protocol, this requirement was necessary due to message size limits in many instant messaging (IM) protocols [1]. Similarly, we can remove the $keyid_A$ and $keyid_B$ serial numbers that are necessary for re-keying in the OTR protocol [15].
Second, and most importantly, we exchanged the traditional Diffie-Hellman key exchange, the long-term key material and the signature algorithm with elliptic curve variants (see Chapter 3 for reference). This allowed us to significantly reduce the amount of data needed to be transferred during the key exchange. Besides, we also improved the security of the DH keys by following the ECRYPT [57] and NIST [63] key size recommendations.
If Alice and Bob know which type of curve and domain parameters are used for the ECDH key exchange, the modified key exchange comprises the steps shown in Protocol 4.2.

## 4.3 Initial authentication with Short Authentication Strings (SAS)

If Alice and Bob connect the very first time and the modified authenticated key exchange (AKE) visible in Protocol 4.2 is applied, they usually do not possess the long-term public keys of each other. As soon as the AKE successfully completes, they both end up with a shared secret and the (alleged) long-term public key of the communication partner, but they do not know for sure whether this information is authentic and no MitM attack has taken place. A look at the AKE reveals that in general the security of the key exchange relies on two types of keys:
The first key type is the ephemeral shared secret key that is established between Alice and Bob by utilizing the Diffie-Hellman key agreement scheme, whereat this type of key agreement only works between two parties [86]. As a result, in case of a MitM attack, Alice would agree on a shared secret key with Eve, the attacker in the middle, and Eve would in turn agree on a

**Protocol 4.2** modified elliptic curve OTR Authenticated Key Exchange (AKE)

1: Alice randomly selects a 512 bit value $d_U$, where $d_U$ is Alice's ECC DH secret key
2: Alice sends SHA256$(Q_U = d_U * G)$ to Bob, where $G$ is the generator point and $Q_U$ is Alice's ECC DH public key
3: Bob randomly selects a 512 bit value $d_V$, where $d_V$ is Bob's ECC DH secret key
4: Bob sends $(Q_V = d_V * G)$ to Bob, where $Q_V$ is Bob's ECC DH public key
5: Alice computes the shared secret $s = d_U * Q_V$
6: Alice sends $(Q_U = d_U * G)$ to Bob, so that Bob can *open* the hash commitment
7: Bob calculates SHA256$(Q_U)$ and verifies that it agrees with the value received in step 2
8: Bob computes the shared secret $s = d_V * Q_U$
9: Alice derives MAC keys $a_1, a_2, b_1, b_2$ and AES keys $a_3, b_3$ from $s$
10: Alice computes $M_A = MAC_{a1}(Q_U, Q_V, v_A, )$, $v_A$ being Alice's ECC public *verify* key
11: Alice computes $X_A = v_a, sign_A(M_A)$, where $sign_A$ is Alice's ECC digital signature
12: Alice sends AES$_{a_3}(X_a), MAC_{a_2}$(AES$_{a_3}(X_a)$) to Bob
13: Bob derives MAC keys $a_1, a_2, b_1, b_2$ and AES keys $a_3, b_3$ from $s$
14: Bob uses $a_2$ to verify $MAC_{a_2}(AES_{a_3}(X_a))$
15: Bob uses $a_3$ to decrypt AES$_{a_3}(X_a)$ and obtains $X_A = v_A, sign_A(M_A)$
16: Bob computes $M_A = MAC_{a1}(Q_U, Q_V, v_A, )$
17: Bob uses $v_A$ to verify $sign_A(M_A)$
18: Bob computes $M_B = MAC_{b_1}(Q_V, Q_U, v_B, )$, $v_B$ being Bob's ECC public *verify* key
19: Bob computes $X_B = v_B, sign_B(M_B)$, where $sign_B$ is Bob's ECC digital signature
20: Bob sends AES$_{b_3}(X_B), MAC_{b_2}(AES_{b_3}(X_B))$ to Alice
21: Alice uses $b_2$ to verify $MAC_{b_2}(AES_{b_3}(X_B))$
22: Alice uses $b_3$ to decrypt AES$_{b_3}(X_B)$ and obtains $v_B, sign_B(M_B)$
23: Alice computes $M_B = MAC_{b_1}(Q_V, Q_U, v_B)$
24: Alice uses $v_B$ to verify $sign_B(M_B)$

different shared secret with Bob. However, if Alice and Bob can compare the shared secret (or the ephemeral public keys that are used to calculate the shared secret) with each other over a secure channel (i.e. *out-of-band*), they can easily detect whether a MitM attack has taken place.
The second type of key is each party's long-term key pair used for signing and signature verification. If a block of data is signed it usually involves the use of a cryptographic hash function. Once the hash value is computed, it is signed by "encrypting" it with a signature algorithm and the private (and thus secret) key of the signer. In case the verifier knows the signer's public key, she can decrypt the hash value and compare it with the one she calculated over the signed data block. If the calculated hash value matches the decrypted received hash value, the received block of data is authentic and the signature is valid. As the signer is the only individual who knows the private key, it is practically infeasible for an attacker to forge a signature (under the assumption that the applied cryptographic primitives are secure and the key lengths are long enough).
Unfortunately, during the initial AKE, Alice and Bob do not possess the long-term public keys of each other and hence they can not verify the authenticity of the key exchange with their digital signatures. As we mentioned, during the Authenticated Key Exchange (AKE) both parties agree

on a shared secret by means of the Diffie Hellman (DH) key exchange. They use this secret to derive a number of keys including a set of MAC keys. In following steps a MAC is calculated over the ephemeral DH public keys ($Q_U, Q_V$) and the long-term public key of the sending party. This MAC proves to the receiving party that the sender was able to calculate the shared secret. By signing the MAC, the receiving party also knows that the sender is in possession of the private long-term key. Hence, to mount a successful MitM attack on the very first AKE, the attacker is forced to use his own long-term signature keys, since otherwise he would not be able to create valid signatures within the AKE protocol. This means however, that the MitM attacker needs to mount a successful attack on all subsequent key exchanges as well since otherwise Alice and Bob would detect the attack.

Therefore, similar to other protocols like SSH [71] (also see Chapter 3), it is of utter importance that the initial key exchange is secure. Once Alice and Bob have securely exchanged their long-term public keys, they can simply compare the other party's public key that is received during the AKE with the stored public key and even more importantly they can check the digital signatures applied in the key exchange. If the keys and the signature match, the key exchange was authentic and no MitM attack has taken place because otherwise, at least the signature of the signed ephemeral DH public keys would not have been valid.

Consequently, Alice and Bob only have two choices to verify the authenticity of the initial AKE: They can either securely exchange their long-term public keys or they can compare their ephemeral shared secret key whereas both needs to be done *out-of-band*.

As the first choice lacks practicability, we focus on the comparison of the ephemeral shared secret key. A practical way to do this is calculate a cryptographic hash value on the key material (i.e. a *fingerprint*) and then compare it out-of-band. As AKE uses a hash commitment in its initial DH exchange, it is in fact not even necessary to compare the whole hash value, but only a fraction of it (i.e. a truncated hash value). To make it easier to compare the truncated hash value, it is possible to use a word list that comprises a special set of words that are clearly distinguishable and understandable over a voice channel. That is instead of directly comparing each byte of the value, each bytes is transformed into a unique word of the word list. Thus for example instead of comparing the fingerprint bytes "0x0b, 0xad, ...", Alice could simply use the words "alone perceptive" whereat the word "alone" represents the byte "0x0b" and "perceptive" represents the byte "0xad". We will have a closer look at this in Chapter 5.

This form of "short authentication" is called Short Authentication String (SAS) [13], proven to be secure [73, 74, 75, 76] and actively used in established protocols like ZRTP [13] as well as in a number of both open-source and commercial secure phones [13, 12, 76, 10]. Due to the hash commitment neither party can influence deterministically the calculated shared secret or the SAS [73]. Also the attacker has only one guess to generate the correct SAS value [13]. If no other way exists to compare the SAS out-of-band, then the same method can be used as in the ZRTP protocol. That is the SAS can be verbally compared by two users by "using their human voices, human ears, and human judgement" [13].

## 4.4 Authentication with Key Continuity Management (KCM)

As we mentioned in the last Section ( 4.3), Alice and Bob can securely verify each others identities as soon as they have the long-term public keys of each other. After the very first initial Authenticated Key Exchange (AKE), in any subsequent AKE execution, they exchange their long-term public keys as well. However, as they *already possess* each others long-term public keys from the very first key exchange, they can do two things:

- They can verify that the received long-term public key matches the *stored* one.
- They can verify the digital signature in the key exchange with the *stored* public key.

Similar to SSH [71], we decided to not only store the long-term public keys from previous communications, but we also decided to *bind* this information to the unique address of each communication partner (i.e. a phone number or a unique network address). This verification can happen completely automatically without any user interaction. In general this procedure is known as *Key Continuity Management (KCM)* [87]. By applying KCM, Alice and Bob have continuity in the way that they know for sure that the person they were previously communicating with is the same person they are communicating with now. This concept has been formalized in what is known as the *duckling security model* [72, 87].
We would like to point out that the use of a Public Key Infrastructure (PKI) would have a similar effect as the AKE combined with KCM, but we intentionally decided to not rely on a PKI for the following two reasons:
First, to use the full potential of a PKI, it is usually required to have online access to the infrastructure (e.g. to retrieve certificates over a Directory Service or to check against the Certificate Revocation List (CRL)). While this might not be a serious problem for devices that are usually online (e.g. a 3G/UMTS phone), it certainly is if communication technologies are used that do not allow online access (e.g. a satellite or radio connection). Yet, the second and main reason is that when utilizing a PKI, one usually has to trust at least one third party, the CA (Certificate Authority), or even a whole bunch of other parties (web of trust). However, with respect to our communication model (Section 4.1), we do not want to trust anyone except for the parties we want to communicate with. That is if Alice calls Bob, she trusts the digital signature of Bob whereas, if Bob is called, he trusts the digital signature of Alice.

## 4.5 Design of the data transfer protocol

Once the Authenticated Key Exchange (AKE) has completed, when transferring encrypted speech or control packets between two parties, we need to maintain the information security properties defined in Section 4.1. At the same time we need to take care of additional requirements such as *replay attack protection* or *synchronization*.
In general, speech and control packets have different demands. While speech packets need to arrive on time (i.e. in *real time*), some amount of packet loss is acceptable as it merely degrades subjective speech quality. In fact cell phone users with bad reception frequently experience the effect of packet loss in their conversations which often manifests as humming or buzzing

noise (depending on the speech compression technology). On the other side there is no strict time constraint for control packets (i.e. *signaling*), but we need to ensure that they are correctly delivered eventually.

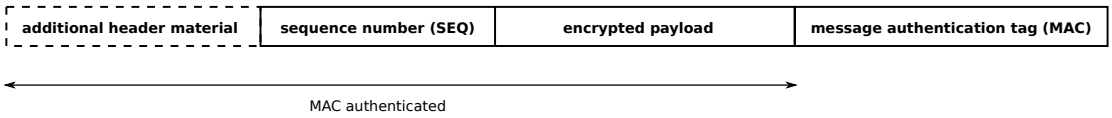


Figure 4.1: general protocol layout for encrypted communication

Figure 4.1 shows the general protocol layout for encrypted communication. The additional header material is required to send the protocol over a network connection. It usually includes fields such as the total length of the packet or the packet type. In terms of cryptographic security of the protocol, it only plays a minor role. The main part of the protocol comprises a sequence number (SEQ), the actual encrypted payload and the message authentication tag (MAC). As can be seen the MAC is computed over all fields of the message, whereat besides message authenticity, it provides message integrity protection as well. The sequence number (SEQ) in combination with the MAC has two important functions.
The first one is to allow detection of packet loss and synchronization between sender and receiver. The receiving side has a local counter that holds the sequence number of the last correctly received and MAC authenticated packet (or 0 at the beginning). A new packet will be only received if it is authentic and the SEQ number of the packet is either equal or greater than the expected sequence number. Thus in case of packet loss, the receiver can easily determine how many packets have been lost as soon as a new authentic packet arrives and re-synchronization with the sender is possible.
The second function is replay attack protection and data freshness. If there would be no sequence number (SEQ) in the packet, an attacker could capture a packet and arbitrarily re-inject (i.e. replay) it at a later point in time. As the MAC field for such a message is authentic, we would accept old packets (i.e. the data would not be fresh) and we would have no way to avoid these attacks. However, with an advancing sequence number (SEQ), each authenticated packet is accepted only once as long as the counter does not overflow. To ensure that this can not happen, at least two solutions exist. One is to make the sequence number (SEQ) field large enough so that for practical application an overflow of the counter will not occur. For instance, if the sequence number field would be 32-bits wide and Alice would send an encrypted speech packet to Bob each 20ms (which for this application is an unrealistically short time period), she could send $2^{32}$ packets before the SEQ number overflows and the communication is prone to a replay attack. Yet this would imply an uninterrupted speech communication duration of $2^{32} * 20$ms which is equal to 23861 hours or 2.72 years. As it is practically extremely unlikely that Alice and Bob will have an uninterrupted conversation taking as long as 2.72 years, even under the worst case assumption that a packet is sent each 20ms, the system would be practically secure.
The other solution to prevent replay attacks in case of a SEQ number overflow is to implement

re-keying. That is, at least whenever the SEQ number gets too high, Alice and Bob will generate new key material so that for message authentication (MAC) a new key will be used. This can either be done by deriving a new key from an existing key or by re-running the Authenticated Key Exchange (AKE) procedure.

## 4.6 Conclusion

We created a security model defining which information security properties our system should provide. As designing a new cryptographic protocol involves a long process of scrutiny and continuous improvement by the cryptographic community, we took the established OTR key exchange as a basis for or own work. OTR is especially suited for our speech communication system, as it provides the basis for all information security properties we defined in our model. We adapted the key exchange by removing *engineering requirements* and utilizing Elliptic Curve Cryptography instead of traditional asymmetric schemes. This way we were able to minimize the bandwidth requirements for the protocol, so that it can be used over low-bandwidth channels. Throughout our adaptation, we took extreme care that the formally proven properties of the original OTR key exchange are not violated. Besides the key exchange, we introduced Short Authentication Strings for initial authentication as well as Key Continuity Management for subsequent authentication purposes. In addition we designed a general data transfer protocol layout that withstands common attacks (e.g. replay attacks) and offers protection for any messages exchanged between Alice and Bob. Now that the theoretical foundation for a secure low-bandwidth communication system is laid, we describe how such an approach can be implemented.

CHAPTER 5

# Implementation

In this section we describe how we implemented a low-bandwidth, portable and secure embedded communication system that relies on the established cryptographic principles and protocols described in Chapter 3 as well as the cryptographic protocol design and security features presented in Chapter 4. Besides its high security, the aim of the implementation was to build a generic end-to-end voice security system that can be used by a broad range of underlying communication media including the ones described in Chapter 2. In general, the system consists of embedded hard- and software, whereas the software can be divided into kernel- and user-land-code.

## 5.1 Hardware

The hardware of the secure communication system comprises Atmel's ARM9 based controller chip AT91SAM9260, an audio codec that functions as a "sound card" and a specialized DSP (Digital Signal Processor) allowing speech compression down to ultra-low bit rates.

### AT91SAM9260 controller and the Olimex SAM9-L9260 board

The AT91SAM9260 is an ARM9 based controller incorporating the ARM926EJ processor that can perform up to 200 MIPS at 180 MHz [2]. Besides the processor, the chip has a high number of useful peripheral features that made it ideally suited for the design of our communication system. A block diagram of the AT91SAM9260 controller chip is depicted in Figure 5.1.

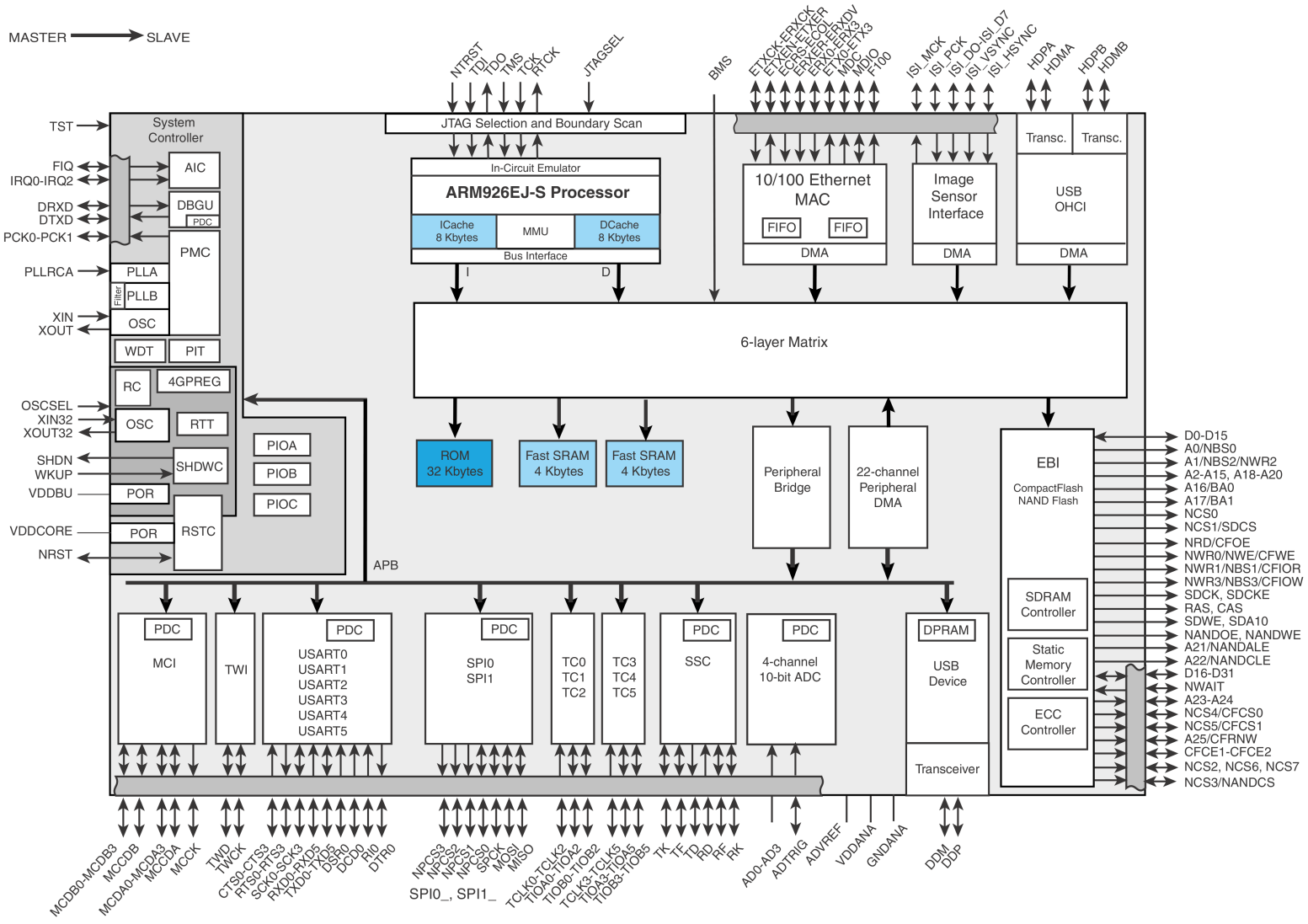


Figure 5.1: Atmel AT91SAM9260 block diagram [2]

The features include:

- a PQFP (plastic quad flat pack) 208 package
- a Power Management Controller (PMC) for low power operation
- support for embedded memories (SDRAM and NAND in particular) through the External Bus Interface (EBI)
- USB 2.0 Full Speed controller allowing host and device ports
- an embedded Ethernet MAC with support for 10/100 Base T
- a programmable oscillators using PLLs (phase locked loops)
- a Synchronous Serial Controller (SSC) with $I^2S$ support
- support for DMA transfers
- support for UARTs, SPI, I2C
- debugging and programming functionality through JTAG and Atmel's SAM-BA (SAM Boot Assistant) ROM bootloader

Usually, large controllers like the AT91SAM9260 come in BGA (Ball Grid Array) packages that are not only hard to solder by hand, but they often require the use of PCBs (printed circuit boards) with more than two layers (which can be costly). As one of our initial design goals was a low-cost system than can be rebuilt without the use of expensive tools, we decided that a PQFP package would be a better choice as it can be soldered by hand and PCBs with no more than two layers should suffice. When building portable systems, low-power operation and power management is an important criterion which is supported by the chip's Power Management Controller (PMC). In addition, the chip supports the use of widely used inexpensive memory types such as SDRAM and Flash, which allows us to run a full-blown Linux distribution on the resulting system at low costs.
USB 2.0 offers the possibility to connect plug-and-play devices such as cell phones. However, for development purposes Ethernet connectivity is very handy as it enables us to mount a (Linux-) software image residing on an NFS server or to connect to the Internet. The programmable oscillators are required in case the audio codec (see Section 5.1) needs to be supplied with a custom clock signal that can be disabled in low-power mode. As the audio codec has to exchange digital audio frames at high speed with the controller chip, the Synchronous Serial Controller (SSC) can be utilized in $I^2S$ mode. The AT91SAM9260 supports DMA transfers which means that these SSC transfers can be done without too much CPU interaction. Other interfaces like UART, SPI (serial peripheral interface) or I2C are important as well as they are required to control the audio codec or exchange data with the speech codec.
For programming and debugging the chip allows JTAG connections. However, as JTAG is usually pretty slow, the proprietary SAM-BA ROM bootloader can be used for programming. If the

chip is powered up with no bootable code in external memories, it automatically launches the SAM-BA bootloader that can be accessed over UART ports or even over USB.
To develop a working communication system prototype (or actually two of them since having only one encrypted speech communication system is rather pointless), we decided to use readily available AT91SAM9260 based evaluation boards. One of these boards (the Olimex SAM9-L9260 development board) is shown in Figure 5.2.

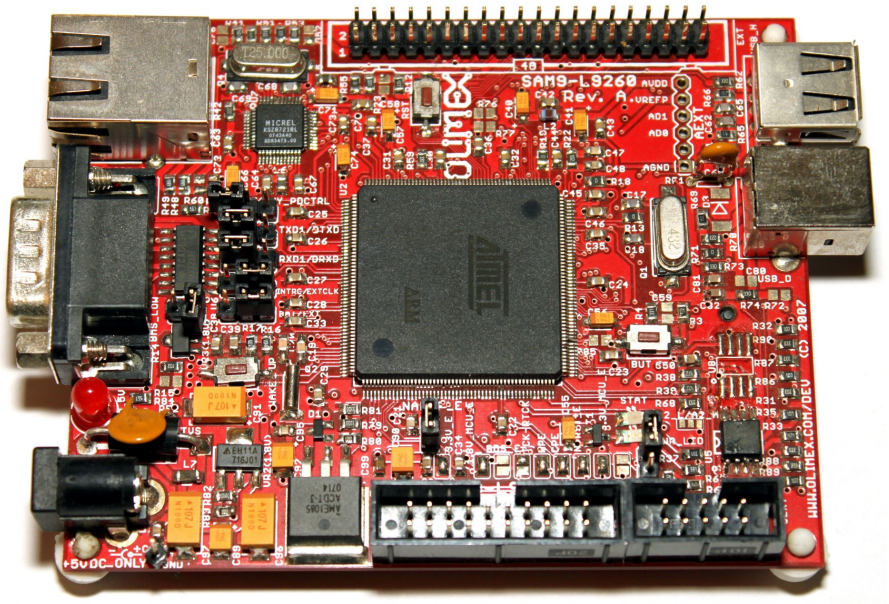


Figure 5.2: Olimex SAM9-L9260 development board

The big advantage of these boards is that they are available at relatively low cost (about 220 EUR per piece) while including most components required to utilize AT91SAM9260's potentials like USB, Ethernet or serial (UART) connectivity. Besides, the board has 64MB DRAM, 512MB NAND Flash and various extension ports required to access SSC, I2C or the UART pins [3]. The full schematic of the board can be seen in Figure 5.3.

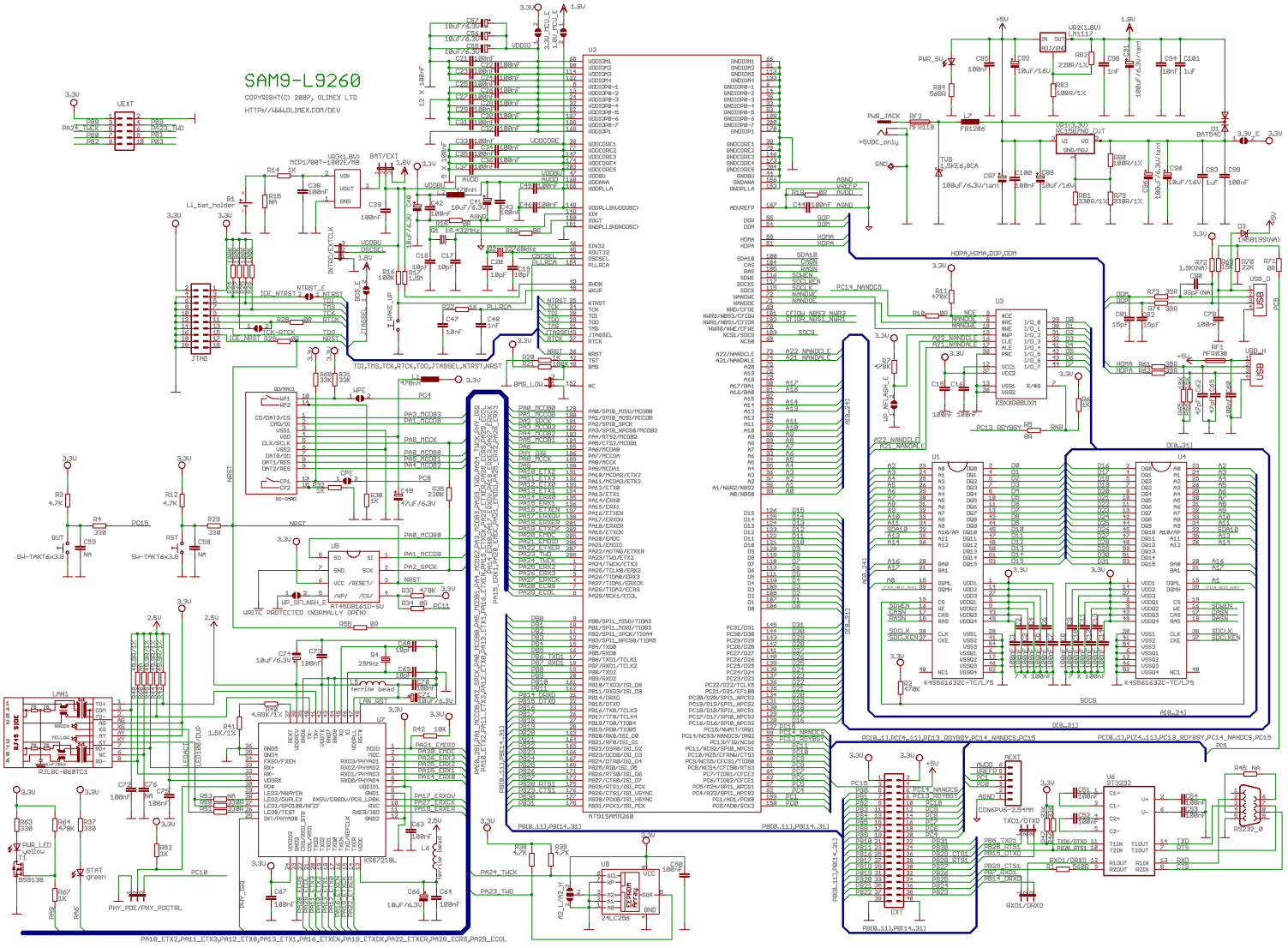


Figure 5.3: Olimex SAM9-L9260 schematic [3]

### Audio Codec

The audio codec is an essential part of our communication system. Comprising a sigma-delta modulation ADC (analog to digital converter) and a DAC (digital to analog converter), it converts analog speech signals coming from the microphone to digital data. Likewise, digital data going into the audio codec is converted back into audible sounds that can be played back on speakers or head phones. The main difference between audio codecs is the ADC/DAC sampling rate used to capture and play back audio as well as the sampling resolution. Essentially, the higher the sampling rate and resolution, the "better" the sound quality will be at the price of a higher amount of data that needs to be processed. The minimum required sampling frequency $f_s$ for a given signal bandwidth $B$ is given by the Nyquist-Shannon theorem visible in Eq. 5.1.

$$f_s > 2B \tag{5.1}$$

According to Rodman, the practical upper frequency limit of standardized digital telephone systems (e.g. G.711) is commonly accepted to be about 3.3 kHz at best [88], whereas in G.711, at a sampling frequency of 8 kHz, the theoretic limit would be 4 kHz. Although the telephone system works well enough for everyday communication, higher sampling rates would improve overall speech quality and intelligibility as "critical elements of speech, the consonants, lie above" that limit [88]. For example, the "high-frequency sound that distinguishes the 's' in 'sailing' from the 'f' in 'failing' occurs between 4 kHz and 14 kHz" [88].
As we have a very tight bandwidth constraint we can not directly transfer the digital audio signal over the communication channel. Even at a sampling frequency of 8 kHz and a relatively low sampling resolution of 8 bits, the resulting bit rate would be as high as $8 * 8 = 64$ kbit/s which is way beyond the maximum capacity of our channel. As we require efficient speech compression in addition to the audio codec, for optimal audio quality, we decided to evaluate two different audio codecs that natively support the audio format (16 bit linear with a sampling rate of 8 KHz) required for the speech compression DSP (see Section 5.1) without the necessity to do format or sampling rate conversions in software.

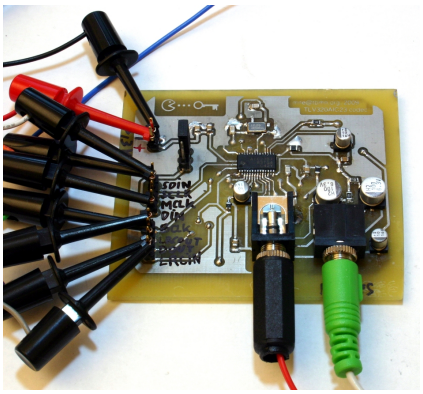

Figure 5.4: TLV320AIC23 test board

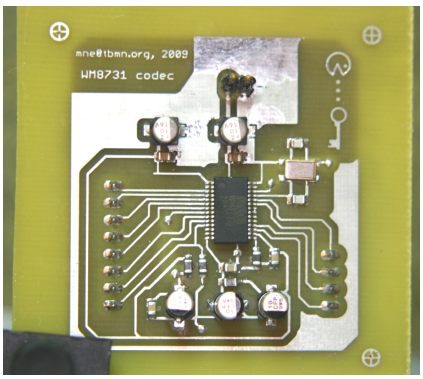


Figure 5.5: WM8731 test board

| connector | AT91SAM9260 pin | TLV320AIC23B pin |
|---|---|---|
| UEXT connector | PA23 (TWD) | codec SDIN |
| UEXT connector | PA24 (TWCK) | codec SCLK |
| EXT connector | PB16 (TK0) | codec BCLK |
| EXT connector | PB17 (TF0) | codec LRCIN |
| EXT connector | PB18 (TD0) | codec DIN |
| EXT connector | PB19 (RD0) | codec DOUT |
| EXT connector | PB21 (RF0) | codec LRCOUT |
| EXT connector | PB30 (PCK0) | codec MCLK (optional) |

Table 5.1: TLV320AIC23B connection to SAM9-L9260 board

The first audio codec we evaluated is the Texas Instruments TLV320AIC23B [89] whereas the second one is the Wolfson Microelectronics WM8731 [90]. For each of these codec chips we constructed test boards visible in Fig. 5.4 and 5.5, respectively.
Essentially, both codecs have the same functionality. They support the 16 bit linear format at 8 kHz, the $I^2S$ frame format compatible with AT91SAM9260's SSC and they have built-in amplifiers for the microphone and headphones. Codec control is possible over SPI (Serial Peripheral Interface) as well as over $I^2C$ (two-wire interface). However, as SPI requires one additional signal line on the PCB (chip select) and seems to be less common with regard to Linux audio drivers, we decided to use $I^2C$. In our test setup, at least from our subjective point of view, the speech quality of the WM8731 seemed to be a bit better than for the TLV320AIC23B codec. However, as the $I^2C$ interface did not work reliably with the WM8731, we decided to use the TLV320AIC23B for the rest of the project.
The schematic for the TLV320AIC23B test board can be seen in Fig. 5.6. We included a jumper (*EXT_CLK*) to either use the on board 12 MHz crystal or an external clock signal coming from one of the AT91SAM9260's programmable oscillators. The advantage of not using the crystal oscillator is current consumption. If the codec is used for recording and playback, it uses at most 26 mA of current [89]. If the codec is disabled and outputs have been turned off, the typical current consumption is 1.5 mA [89]. However, if the clock signal is completely turned off (e.g. by means of the programmable oscillator), the current consumption would be as low as 0.01 mA [89].
During evaluation it turned out that with the 12 MHz crystal, standard sampling rates like 8 kHz or 44.1 kHz are not possible. Instead, due to the crystal frequency, the nearest possible sampling rates are 8.021 kHz and 44.117 kHz, respectively [89]. However, if at one system sound is captured at a nominal sampling frequency of 8 kHz (which is in fact 8.021 kHz) and the sound is played back at another system with an exact sampling frequency of 8 kHz, it causes an unwanted pitch of the sound. To avoid the issue, in the board visible in Fig. 5.4 we had to exchange the crystal with a 12.288 MHz one.
To use the TLV320AIC23B test board with the Olimex SAM9-L9260 evaluation board, we connected the boards as illustrated in Table 5.1.

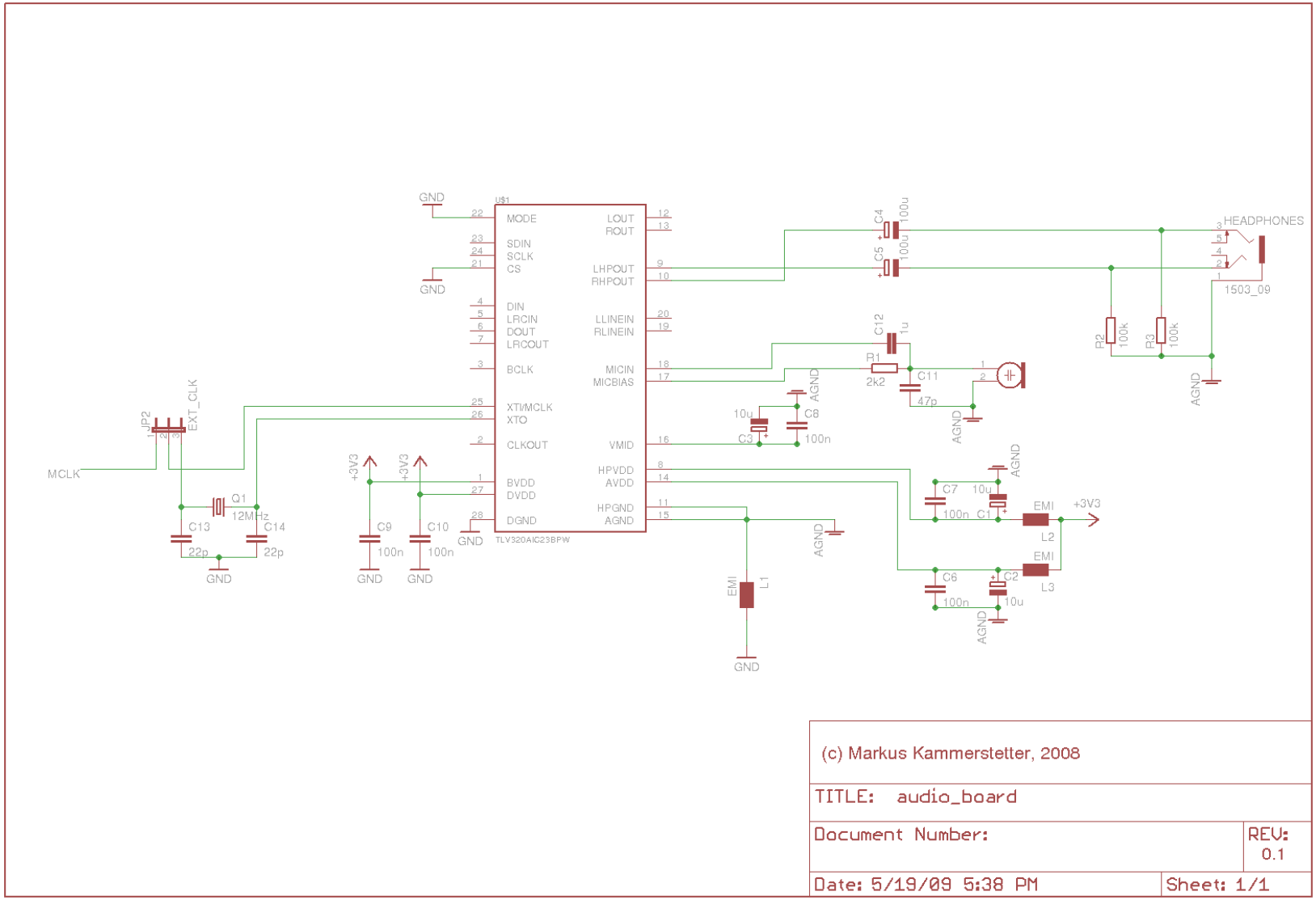


Figure 5.6: TLV320AIC23B test board schematic

### Speech Compression DSP

Besides the cryptographic implementation, the speech compression codec is one of the most critical parts in our communication system. Without a proper codec ultra low bandwidth communication would not be possible at all. Finding a speech compression codec that works well in an ultra low bandwidth environment (9.6 kbaud/s and below), is however not a trivial task. Assuming that for serial communication the 8N1 data format is used, for each byte of payload one additional start- and stop-bit needs to be transferred. Thus, at 9.6 kbaud/s the maximum possible speech bit rate would be $\frac{9.6*8}{10} = 7.68$ kbit/s. But as additional header material such as Message Authentication Codes (MACs) needs to be transported as well, the bandwidth that can be used for speech is significantly below 9.6 kbit/s and more likely to be in the $2 - 4$ kbit/s range. (We will discuss this in more detail in Section 5.2 and Chapter 6). Due to the strict bandwidth limits common low bandwidth speech codecs like GSM or iLBC (Internet Low Bit Rate Codec) can not be used and specialized speech codecs like CELP (Code-excited linear prediction) [91], LPC-10 (Linear predictive coding) [91], MELP (Mixed Excitation Linear Prediction) [91] or AMBE (Advanced Multi-Band Excitation) [92, 91] need to be considered.
We compared codec speech samples at bit rates up to 4.8 kbit/s with the result that the AMBE (Advanced Multi-Band Excitation) codec outperforms the mentioned codecs in terms of speech quality and corresponding bandwidth requirements. Similar results have been found in evaluations conducted by Neto et al. [93] and other sources as well [91, 94, 95, 96, 97]. As, when we started to implement the system, no suitable royalty free codecs like Codec2 [98] existed and AMBE was available in a ready to use DSP with additional useful features like echo cancellation or comfort noise generation, we chose to utilize Digital Voice System's AMBE-3000 chip [92].

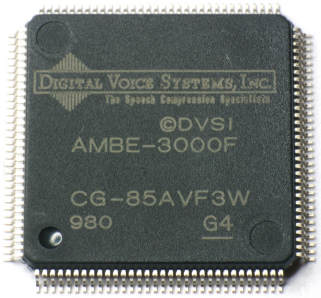


Figure 5.7: AMBE-3000F speech compression DSP

The DVSinc. AMBE-3000F speech compression DSP depicted in Fig. 5.7 uses the Texas Instruments TMS320F2811 DSP core to implement the Advanced Multi-Band Excitation (AMBE) proprietary speech coding algorithm [92]. Is supports variable data rates ranging from 2.3 to 9.6 kbits/s, whereat a variable amount of FEC (Forward Error Correction) can be selected. The chip can operate in two different modes. In *codec mode* two different physical interfaces are used at the same time. On one interface (either SPI or McBSP) the audio codec is directly connected to the chip, whereas on the other interface (e.g. UART or parallel), the compressed speech data goes in and out. While this mode allows a less complex hardware implementation it has the drawback that the audio codec can not be used independently by the AT91SAM9260 controller.

This is why we chose to use *packet mode*. In packet mode only one physical interface (UART, McBSP or parallel) is used and uncompressed voice as well as compressed data is transferred in the form of data packets. To achieve the maximum voice quality it is recommended that a 16-bit linear audio codec is used with the AMBE-3000 DSP while the sampling rate needs to be $8$ kHz [92]. During full-duplex operation, the DSP operates on $160 \pm 4$ codec samples at a time. At a sampling frequency of $8$ kHz, one period is $\frac{1000000\mu s}{8000Hz} = 125\mu s$, hence a speech packet comprising 160 samples needs to be transferred between the AT91SAM9260 controller and the DSP each $160 * 125\mu s = 20ms$. Likewise, compressed speech packets need to be transferred each $20ms$ as well. To evaluate which interfaces and data transfer modes work best for our communication system, we implemented a prototyping board visible in Fig. 5.8.

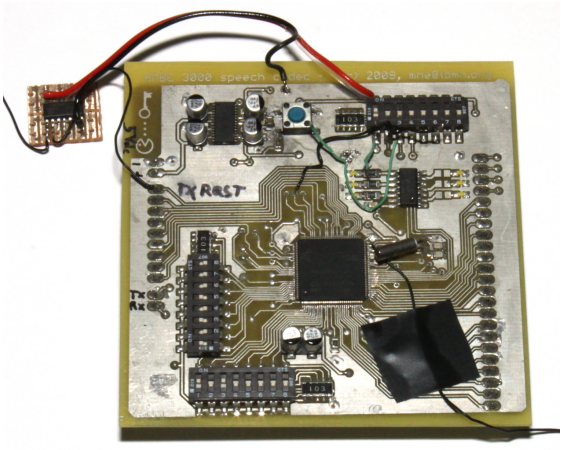

Figure 5.8: AMBE-3000F prototyping board

The prototype showed that using the DSP's UART at $460$ kbaud/s in packet mode with hardware (RTS/CTS) flow control works well in conjunction with UART0 of the AT91SAM9260 controller. However, in order to use AMBE-3000's *RTS* and *TX_RQST* pins, an additional inverter (visible in the upper left corner) for the TX_RQST signal was necessary. To use the AMBE-3000 prototyping board with the Olimex SAM9-L9260 evaluation board, we connected the boards as illustrated in Table 5.2. The schematic of the board is visible in Fig. 5.9.

| SAM9-L9260 connector | AT91SAM9260 pin | AMBE-3000 pin |
|---|---|---|
| EXT connector | PB4 (TXD0) | UART_RX |
| EXT connector | PB5 (RXD0) | UART_TX |
| EXT connector | PB27 (CTS0) | RTS |
| EXT connector | PB26 (RTS0) | inverted TX_RQST |

Table 5.2: AMBE-3000 prototyping board connection to SAM9-L9260 board

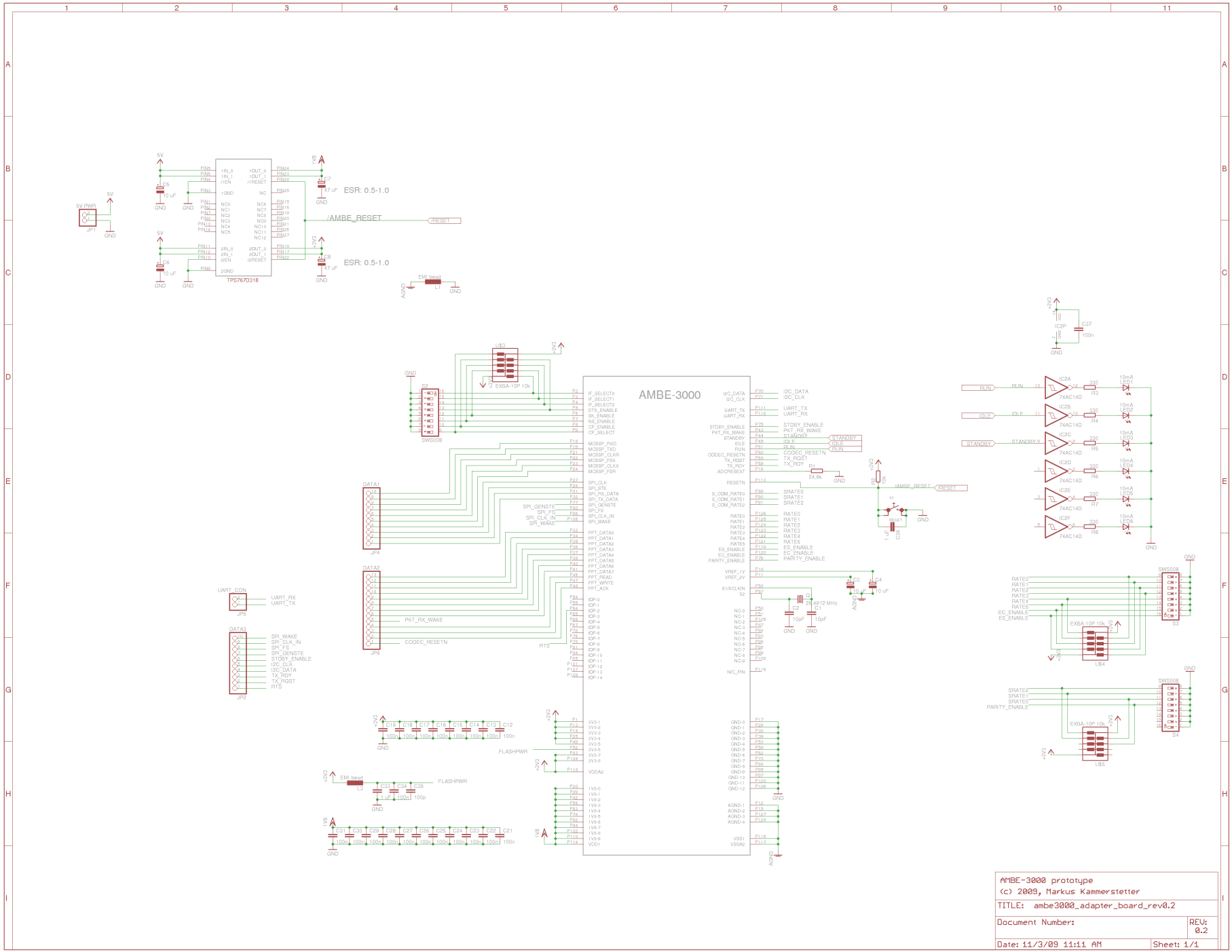


Figure 5.9: AMBE-3000 prototyping board schematic

### Combined audio codec and speech compression board

After evaluating the audio codec and the speech compression test boards, we created a combined board that can be easily connected to the Olimex SAM9-L9260 board. In contrast to the test boards, the changes that turned out to be necessary during evaluation are already included and breakout connections that are not required for operation were removed. Besides, the amount of required space was significantly reduced. The PCB was professionally produced in a board house and soldered in a reflow oven. The finished board can be seen in Fig. 5.10 while the completed encrypted speech communication hardware is depicted in Fig. 5.11

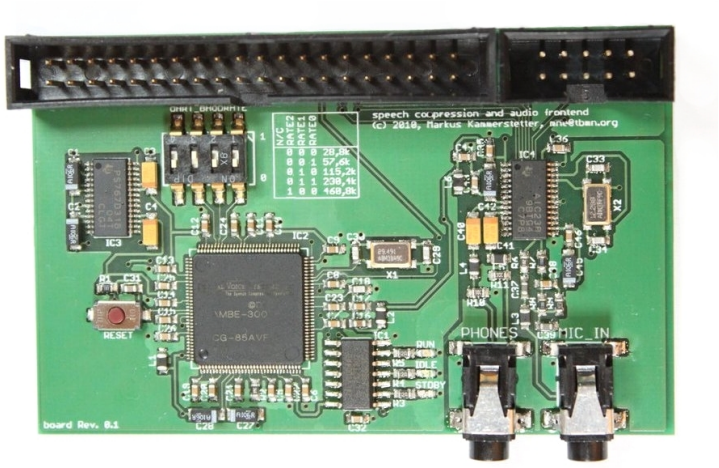


Figure 5.10: combined audio codec and speech compression board

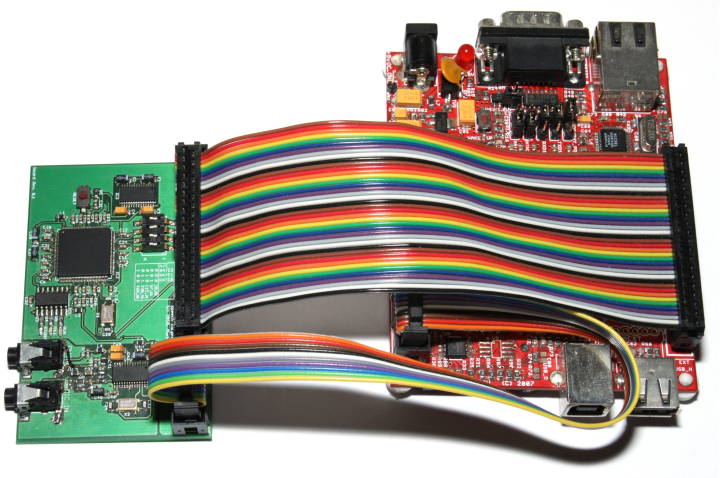

Figure 5.11: functional encrypted speech communication hardware

## 5.2 Software

We decided to use the Linux operating system as the software basis for our secure communication system as it brings a number of key advantages. The Linux kernel supports the AT91 architecture and the AT91SAM9260 controller with most of its embedded peripherals such as the SSC (Synchronous Serial Controller), UART, USB or the embedded Ethernet MAC. In addition, it includes the ALSA (Advanced Linux Sound Architecture) infrastructure together with the ALSA System on a Chip (ASoC) subsystem, a framework for embedded audio with support for the TLV320AIC23 audio codec, making it a good choice for our application. Besides the Linux kernel, the Linux (GNU) user-land allows us to leverage a vast amount of practically proven and tested software such as development tools, (e.g. cryptographic) libraries or other useful applications. This way we can not only avoid re-inventing existing solutions, but the resulting system will also become feature rich and powerful in terms of applicability. Rather than building a highly specialized system that can only be applied for encrypted speech communication over a small number of communication media and devices, the implementation can be combined with existing software to cover a broad range of different media and devices. The resulting system is so versatile, that encrypted speech communication is merely one application among others. For instance, by leveraging existing Linux applications, our system can be easily used as digital speech recorder, MP3 player, Internet radio or real-time speech compression device. Reaching this high amount of versatility was possible by implementing our system in a generic enough way, so that it can be integrated with existing Linux kernel- and user-space applications. However, before we could start to use the full potential of Linux, we needed to port the Linux kernel and various bootloader stages and to our hardware platform (see 5.1).

### Bootloader stages

Fig. 5.12 shows the memory mapping of the AT91SAM9260 controller. As soon as the controller is powered up, it will start to execute code from the boot memory at *0x00000000*. However, the 64 KByte boot memory region at *0x00000000 - 0x00010000* is no physical memory, but depending on the configuration (BMS and REMAP [2]) different types of memory (and thus also different memory regions) can be mapped to that address. Essentially, we can either map the internal SAM-BA (SAM Boot Assistant) ROM bootloader to that region or we can use an external memory that is connected to the EBI (External Bus Interface) and selectable with *NCS0*. On the SAM9-L9260 board we have two types of memory that we can use for booting. The first type is a 2 MByte serial DataFlash memory (Atmel *AT45DB161D* 16 MBit) while the other type is a 512 MByte NAND flash memory (Samsung K9F4G08U0M 512 M x 8 bits). However, as both memories are not connected to *NCS*0 and they lack a parallel memory interface that could be directly accessed over the EBI, on our hardware platform we always need to boot into the SAM-BA ROM bootloader.

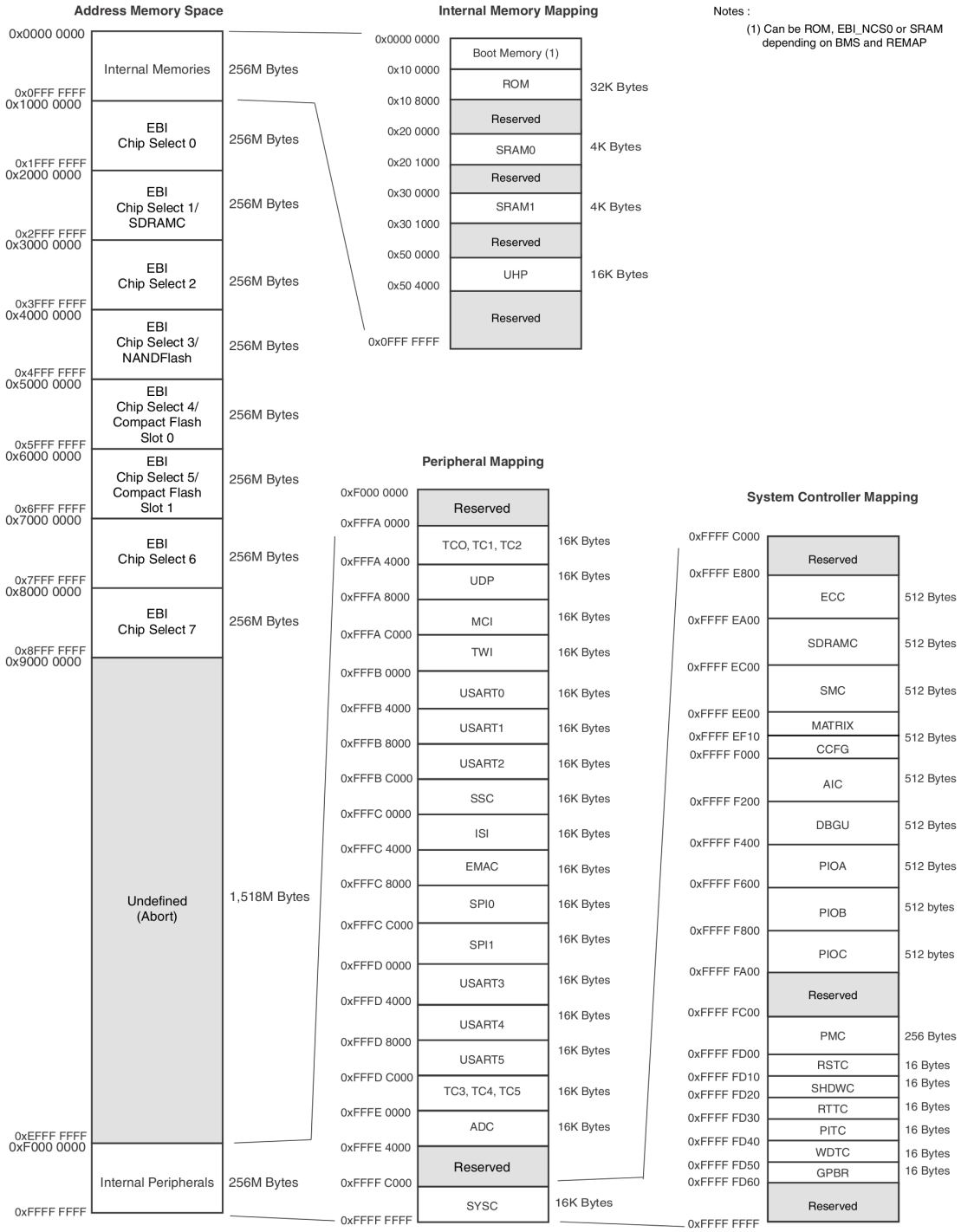


Figure 5.12: AT91SAM9260 memory mapping [2]

When powering up the AT91SAM9260, the CPU runs at reduced clock speed (32.768 kHz) and the ROM bootloader tries to determine whether a valid boot program is in the serial DataFlash or NAND flash memory by checking whether a sequence of eight valid ARM exception vectors can be found [2]. If no valid boot program is found, the internal SAM-BA boot program takes control, which allows to access and program the device over USB (for example by utilizing the SAM Boot Assistant Linux port software tool [99]). Otherwise the content (up to 4 KBytes) is copied to the internal SRAM (*SRAM*0), the SRAM is remapped to the boot memory region and the copied boot program is executed. Therefore, the SAM-BA ROM bootloader denotes the *first bootloader stage*.
When executing the code in the SRAM, the controller still runs at the slow clock and the maximum size of the code being executed is limited to the 4 KBytes of the SRAM. Thus, before being able to run our main application (i.e. the Linux kernel), a number of low-level hardware initializations need to be done. This is the task of the AT91Bootstrap bootloader [100, 101], the *second bootloader stage*. Among the most basic things it sets up is the SD-RAM controller and the system master clock. Since we opted for the large NAND flash memory (in relation to the smaller DataFlash memory) and the NAND flash memory chip (i.e. the Samsung K9F4G08U0M) was not supported by the AT91Bootstrap bootloader, we were required to make some changes in the code.
Throughout the project we used a Buildroot [102] generated cross-compilation environment to build uClibc [103] based EABI (embedded-application binary interface) binaries compatible with the ARM9 architecture of the AT91SAM9260.
While AT91Bootstrap would allow us to boot into a Linux kernel, it has the drawback that valuable features such as flash memory programming or network booting are not available. As these features are essential during application development, we decided to use the U-Boot [104] bootloader, the *third bootloader stage*. According to Yaghmour et al. it is "arguably the richest, most flexible, and most actively developed open source embedded bootloader available [...] capable of booting a kernel through TFTP over a network connection, [...] from USB, and from a wide variety of flash devices" [105]. To make it work with our hardware platform, we had to port the U-Boot bootloader.
After setting up *dhcpd*, *tftpd* and *NFS* servers on a development machine, we were finally able to run the Linux kernel and boot into a NFS-mounted file system. We used the u-boot script shown in Listing 5.1 to boot the system.

**Listing 5.1: U-Boot boot script**

```
setenv ethaddr 00:de:ad:be:ef:00
setenv loadaddr 0x21500000
setenv autoload no
setenv bootcmd 'dhcp; tftp; bootm 0x21500000'
setenv bootargs mem=64M console=ttyS0,115200, noinitrd root=/dev/nfs rw
    nfsroot=192.168.1.105:/export/buildroot ip=dhcp
saveenv
dhcp
tftp
bootm 0x21500000
```

### Porting the Linux kernel

In order to run the Linux kernel on our target system we needed to port the kernel. As a starting point we used kernel *2.6.28* and patched it with the AT91 maintainer patch set [106]. To get the latest fixes we also forward ported the Linux4SAM experimental patch set for *kernel 2.6.27* [107] to kernel *2.6.28*. One of the key improvements of this patch set was related to the Ethernet MACB driver. In the errata section of the AT91SAM9260 controller it is mentioned that the Ethernet MACB FIFO arbitration may lead to TX underruns [2], potentially resulting in slow Ethernet data transfers. In our initial test setup this led to the Ethernet transfers being so slow, that mounting the root file-system over NFS was infeasible. However, as the controller's internal 4 KByte SRAM is no longer used as soon as the system has initialized the SD-RAM, we were able to fix the underrun problem with the forward ported experimental patch set by utilizing the internal SRAM as MACB TX buffer.
We achieved the actual port of the Linux kernel to our hardware setup by implementing a so-called *"board"*-file in *arch/arm/mach-at91*. Essentially, this file defines a new *machine* where the Linux kernel should run on. Once the machine has a registered *machine type number* in *include/asm-arm/mach-types.h*, the *MACHINE_START* preprocessor macro shown in Listing 5.2 can be defined. Among other fields, *struct machine_desc* contains the function callback pointers `*map_io()` and `*init_machine()`.
The **`*map_io()`** callback is called indirectly at the very beginning of C kernel code execution by `start_kernel()` in *init/main.c*. It is responsible for initializing the CPU, mapping the I/O address space as well as registering and initializing various clocks like the system clock or clocks required for peripherals.
The **`*init_machine()`** callback is called at the end of kernel initialization. It is responsible for registering all platform devices and it also calls the initialization code for the peripherals that we intend to use. That is, for example we call

```
at91_add_device_ssc(AT91SAM9260_ID_SSC, ATMEL_SSC_TX | ATMEL_SSC_RX)
```

to initialize the SSC (Synchronous Serial Controller) whereat with

```
ATMEL_SSC_TX | ATMEL_SSC_RX
```

we specify that all SSC pins required for I2S *(TK, TF, TD, RK, RF, RD)* should be used. In a similar way we initialized other peripherals on the board like the Ethernet MAC, SPI, I2C, UART or the NAND flash controller.
Once the peripheral devices are registered and initialized, the corresponding device drivers can be loaded to utilize the devices. Accessing the devices is possible through the memory mapped architecture (see 5.12) of the AT91SAM9260 controller [2]. For example, if the SSC needs to be accessed by the driver (*drivers/misc/atmel-ssc.c*), it can do so by addressing the necessary SSC configuration registers using the physical memory region at *0xfffbc000* [2].

**Listing 5.2: MACHINE_START macro defined in *arch/arm/include/asm/mach/arch.h***

```
struct machine_desc {
        /*
         * Note! The first four elements are used
         * by assembler code in head.S, head-common.S
         */
        unsigned int            nr;             /* architecture number  */
        unsigned int            phys_io;        /* start of physical io */
        unsigned int            io_pg_offst;    /* byte offset for io
                                                 * page tabe entry      */

        const char              *name;          /* architecture name    */
        unsigned long           boot_params;    /* tagged list          */

        unsigned int            video_start;    /* start of video RAM   */
        unsigned int            video_end;      /* end of video RAM     */

        unsigned int            reserve_lp0 :1; /* never has lp0        */
        unsigned int            reserve_lp1 :1; /* never has lp1        */
        unsigned int            reserve_lp2 :1; /* never has lp2        */
        unsigned int            soft_reboot :1; /* soft reboot          */
        void                    (*fixup)(struct machine_desc *,
                                         struct tag *, char **,
                                         struct meminfo *);
        void                    (*map_io)(void);/* IO mapping function  */
        void                    (*init_irq)(void);
        struct sys_timer        *timer;         /* system tick timer    */
        void                    (*init_machine)(void);
};

/*
 * Set of macros to define architecture features.  This is built into
 * a table by the linker.
 */
#define MACHINE_START(_type,_name)                      \
static const struct machine_desc __mach_desc_##_type   \
 __used                                                 \
 __attribute__((__section__(                ))) = {     \
        .nr             = MACH_TYPE_##_type,            \
        .name           = _name,

#define MACHINE_END                             \
};
```

### ASoC audio driver implementation

Although the ASoC (ALSA System on Chip) subsystem in the patched kernel *2.6.28* already supported the TLV30AIC23B codec, there was no sound driver for our hardware and we were required to write a new one. Usually, in regular ALSA sound card drivers the driver code always includes all code necessary to communicate with the sound card hardware. In case there are multiple sound cards from different manufacturers that all use the same audio codec, the driver code responsible for the audio codec would usually be re-implemented in each driver resulting in unwanted *code duplication*. On embedded systems the situation is even worse, as a full driver implementation would be required for each machine (i.e. board) and each platform leading to a plethora of different independent drivers that actually have a very similar functionality. The ASoC subsystem is designed to address these issues by splitting up embedded audio systems into three components [108]:

1. codec driver
2. platform driver
3. machine driver

The **codec driver** is a platform-independent driver providing audio capture and playback functionality. Comprising audio controls and interface capabilities, it includes a number of codec I/O functions and DAPM (Dynamic Audio Power Management) capabilities. In our case there is a ready-to-use driver for the TLV320AIC23B audio codec in *sound/soc/codecs/tlv320aic23.c*. The codec DAI (Digital Audio Interface) describes the type and configuration of the physical digital audio interface whereas the PCM configuration defines which audio capabilities the codec has (e.g. sampling rates and formats). To make the codec work in our environment, we need to use the SSC in I2S mode as DAI whereas for codec I/O control we use the I2C bus.
The **platform driver** contains the audio DMA engine and DAI drivers (e.g. for I2S) for each SoC CPU. To use I2S with the Atmel SSC, we utilized the "ALSA SoC ATMEL SSC Audio Layer Platform driver" available in *sound/soc/atmel/atmel_ssc_dai.c*.
The **machine driver** handles machine- (i.e. board-) specific controls and audio events. It can be seen as the *glue* sticking together the platform and the codec driver. As no previous drivers existed for our encrypted speech communication hardware, we implemented a new machine driver allowing us to use the custom TLV320AIC23B based sound card design (see Section 5.1). Our machine driver sets up the SSC in I2S slave mode. The codec is responsible to generate the required BCLK clock and the LRCIN/LRCOUT frame synchronization signals.
Since our driver implementation acts as a regular ALSA sound card, we can use arbitrary Linux sound applications with our hardware setup. For testing purposes we were able to playback MP3 files and listen to Internet radio in decent quality. Also audio recording and full duplex operation (i.e. audio capture and playback at the same time) worked well.

### AMBE speech compression I/O plugin

As described in Section 5.1, the AMBE-3000 speech compression DSP is connected to *UART0* of the AT91SAM9260 controller which allows us to take advantage of hardware flow control

capabilities. In this setup the chip can be used in full duplex *packet mode* to compress and decompress speech at the same time. To achieve the best possible amount of speech quality, the uncompressed speech samples supplied to the DSP need to have a sampling rate of 8 kHz while the audio format is 16-bit signed little endian.
For speech compression, uncompressed speech packets consisting of 160 speech samples are sent to the DSP each 20ms. However, as we need to wait until 20ms of audio was captured from the microphone, this will also introduce additional latency. The AMBE compression algorithm is executed on the DSP and after the processing delay, the chip returns a compressed *channel packet*. Likewise, for speech decompression, a channel packet is sent to the DSP each 20ms, the AMBE decompression algorithm is executed and an uncompressed speech packet is returned after the processing delay.

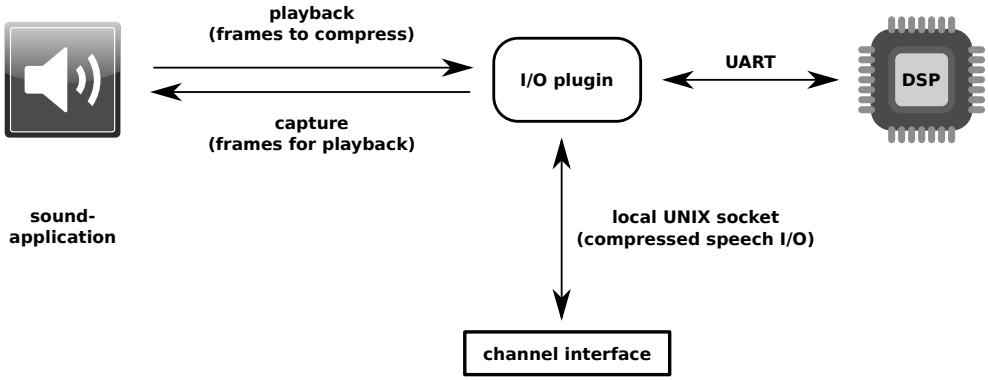


Figure 5.13: speech compression I/O plugin concept

To transparently support AMBE speech compression and decompression for ALSA applications, we decided to implement an ALSA I/O plugin. The plugin acts in a similar way as a sound card. It can be opened for audio capture and playback, whereat both is possible at the same time to support full-duplex operation. The concept is depicted in Fig. 5.13. In playback mode the sound application sends audio samples to the plugin, but rather than "playing them back" (e.g. through a pair of speakers), all samples are sent to the DSP. Similarly, in capture mode, the sound application receives audio samples from the plugin, which in turn receives those samples from the DSP.
Besides, the plugin allows to access compressed speech packets over a local UNIX socket. All channel packets sent to the socket are passed on to the DSP and, vice versa, all channel packets from the DSP can be read through the socket. During operation, speech being played back through the plugin is made available in compressed form on the UNIX socket, whereas compressed channel packets sent to the socket is made available in decompressed form on the audio capture interface.

Implementing the I/O plugin was one of the most challenging tasks throughout the project. One of the major issues we had to tackle was to optimize the plugin communication design so that a continuous real-time audio stream could be guaranteed. This led to a multi-threaded plugin design with signaling pipes for inter-thread flow control (see Fig. 5.14).

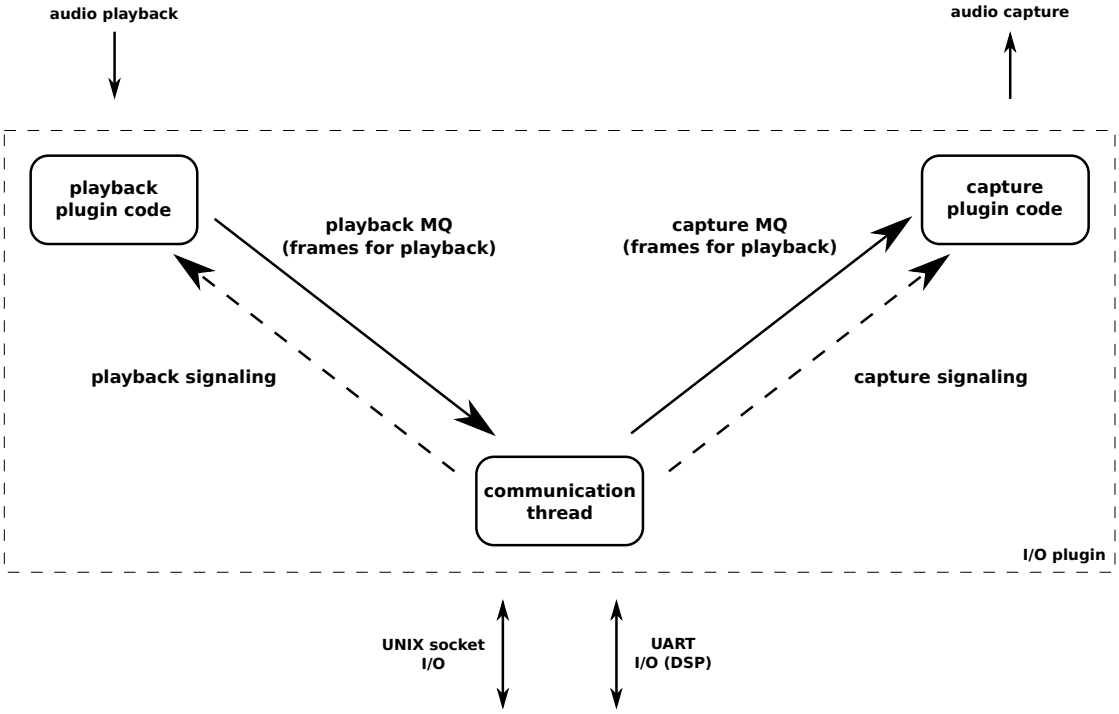


Figure 5.14: speech compression I/O plugin design

ALSA plugins are implemented as shared libraries. If a sound application opens the plugin in playback mode, the plugin shared object will be loaded into the process space of the sound application and relevant functions within the plugin are called. In this case, the playback plugin code spawns the communication thread. Otherwise, if the application opens the plugin in capture mode first, the capture plugin code has to spawn the shared communication thread. For obvious reasons, synchronization between the capture and the playback code is necessary to avoid spawning the shared communication thread multiple times.
Once the communication thread is running, it will initialize the AMBE3000 DSP over the UART connection and packet processing can start. The DSP has internal FIFOs "large enough to accommodate up to two speech packets and two channel packets" [92] at the same time. In the optimal case, while the DSP processes one packet, it already receives the next packet so that no additional delays result from the UART packet transfers. Essentially, the goal is to always keep the internal FIFOs filled.

The AMBE-3000 chip uses hardware flow control to signal whether new data is available for reading or new data can be sent for processing. To achieve a design enabling real-time communication, we forward this information through the signaling pipes up to the ALSA application itself (Fig. 5.14). If the DSP is ready to accept a new chunk of 160 speech samples, it will be signaled to the ALSA application that it can can immediately transfer the speech chunk from its internal buffer to the DSP. Likewise, whenever the DSP has decompressed a channel packet, the ALSA application will be notified that a new speech chunk is ready to be moved to the internal capture buffer. The actual data transfers between the communication thread and the playback/capture code utilize IPC message queues.

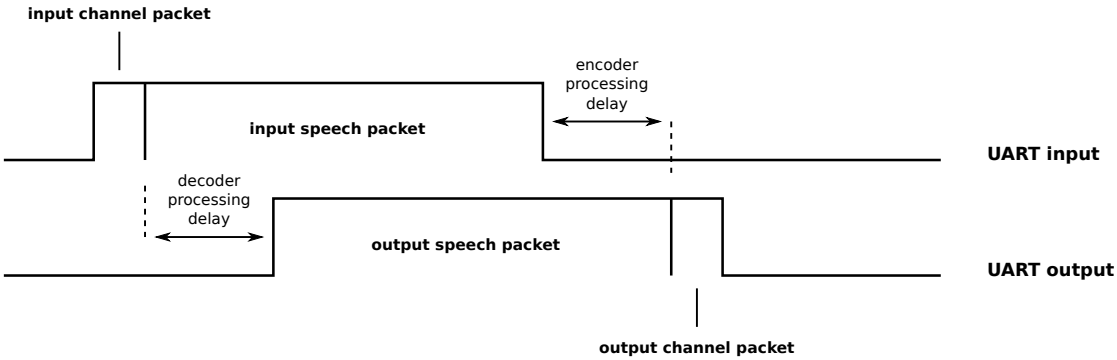


Figure 5.15: AMBE-3000 packet timing

In addition to flow control, combining channel and speech packets in UART transfers is essential as depicted in Fig. 5.15. Unfortunately, speech and channel packets may not necessarily arrive at the communication thread at the same time. If both, playback and capture, are active, at one time, the communication thread may receive a chunk from the playback plugin code while it may receive a compressed speech packet on the UNIX socket at another time. A naive implementation would be to just send speech and channel packets to the chip as soon as they arrive at the communication thread. This could, however, lead to the case that transfer delays will effectively double. For example, if a speech packet arrives and a channel packets is received shortly after that, there would be two separate (slow) data transfers for each of those packets. We mitigated this problem by implementing a packet scheduler that basically initiates a data transfer whenever a new speech packet is available from the ALSA application. If, at that time, also a channel packet is in the queue, it will be added to the packet that is transferred to the DSP. This way each data transfer contains the maximum amount of data available and transfer delays are minimized. Besides sending packets, a decision has to be made when response packets should be received from the UART. While the underlying hardware may receive packets at any time, the communication thread needs to know whether a packet can be read from the kernel buffer. Reading while no data is in the receive buffer would cause unnecessary or even critical additional delays. Our packet scheduler tackles the problem by utilizing the *poll(2)* system call. Due to the high amount of complexity involved with the design of the plugin, the overall plugin implementation together with the AMBE-3000 driver code resulted in the plugin being one of the largest implementation parts with over 7000 lines of C source code.

### Encrypted speech communication application

The encrypted speech communication application is a multi threaded application to place and receive secure calls. It comprises a number of threads responsible for different tasks:

- sound processing thread
- speech plugin accept thread
- secure transmission thread
- secure reception (main program)

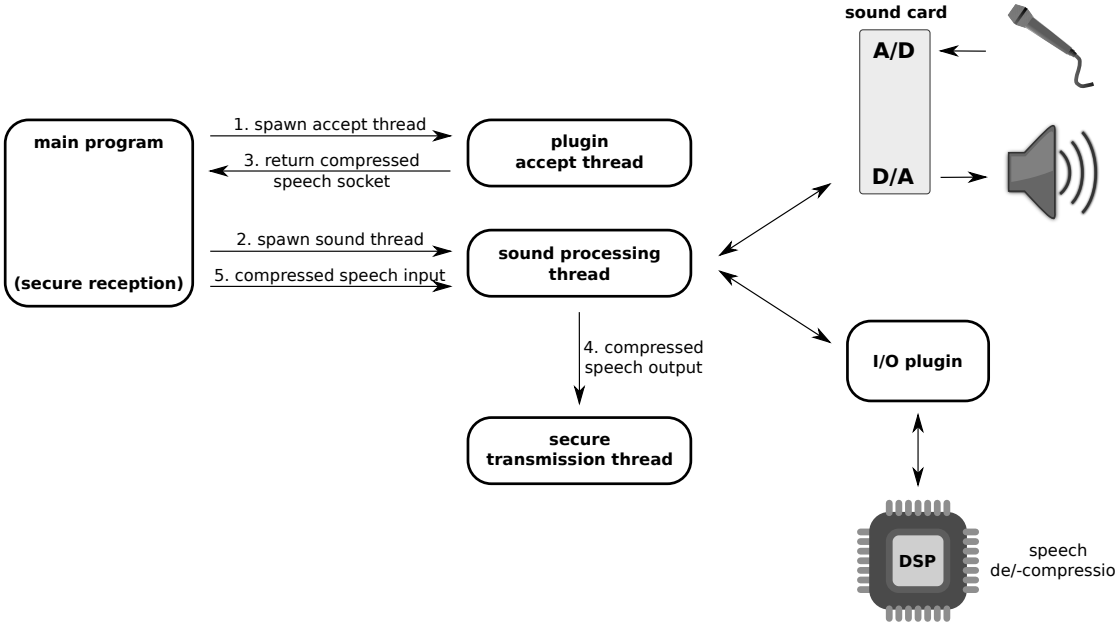


Figure 5.16: thread overview

The **sound processing thread** opens both a sound device (i.e. the TLV320AIC23B sound card) and the speech compression I/O plugin for capture and playback at the same time (i.e. in full-duplex mode). Using an event triggered architecture based on *poll(2)*, it continuously transfers audio frames between the sound card and the plugin. Speech from the microphone is sent to the AMBE speech compression plugin while received decompressed speech is played back (e.g. on headphones). Due to the transparent design of the speech compression plugin, arbitrary sound devices could be used. Instead of utilizing the TLV320AIC23B sound card, one could also use a Bluetooth headset or the handsfree set of a car. The only prerequisite for this is that the corresponding hardware is supported by ALSA and the Linux kernel. Also wireless technologies like Bluetooth bring their own security implications, thus in the end it is the user who can choose the tradeoff between security and usability.

The **speech plugin accept thread** is an engineering requirement. As soon as the sound processing thread is started, the AMBE speech compression plugin will try to connect to a listening UNIX socket for compressed speech I/O communication. However, from the perspective of a single threaded process, spawning the sound processing thread and accepting the incoming socket connection has to happen concurrently. If in the code of the process one would first call *accept(2)* and then spawn the thread, the process would wait forever in the *accept(2)* call as there would be no incoming connection since the sound thread does not exist yet. Otherwise first creating the sound thread and then calling *accept(2)* would not work either, as in this case, at the time when the sound thread starts, no listening socket might exist yet. While we could cope with this situation by applying IPC synchronization mechanisms, we decided that using a separate thread to accept the connection "in the background" requires less effort. During experimental evaluation it turned out that the approach works well.
Sending and receiving encrypted speech frames happens asynchronously and independently from each other. This allowed us to create an event triggered architecture and a less complex implementation. Whenever speech from the microphone is compressed by the plugin, it is immediately queued, combined with previous packets and sent over the encrypted channel as soon as the amount of combined packet reaches a preset threshold. All these operations are handled by the **secure transmission thread**.
On the other side the tasks necessary for **secure reception** are implemented in the main program itself. Whenever packets are received over the secure channel, the packets are decompressed and buffered until they are finally played back. The overall thread concept is visible in Fig. 5.16.

However, before compressed speech frames can be exchanged between Alice and Bob, a secure channel needs to be set up in the way we described in Chapters 3 and 4. Essentially, to do so, we implemented the following:

**AKE**
: Utilizing the established and well tested libtomcrypt [109] library, we implemented the Authenticated Key Exchange as described in Section 4.2.

**SAS**
: We implemented Short Authentication Strings as described in Section 4.3 by utilizing the PGP biometric word list [110]. It comprises a specialized list of words "for conveying data bytes in a clear unambiguous way via a voice channel". These words were "carefully chosen for their phonetic distinctiveness, using genetic algorithms to select lists of words that had optimum separations in phoneme space" [110].

**KCM**
: We implemented Key Continuity Management as described in Section 4.4 by storing the other parties address (e.g. a unique network address) together with the corresponding long-term ECDSA public key in a "known parties" file. Essentially, this is very similar to SSH's *known_hosts* [71] file.

**transport protocol**
: The transport protocol closely resembles the structure defined in Section 4.5, whereat

two different transport mechanisms are supported. On one hand we support *TLV (Type,Length,Value)* triplets to exchange call management information with the communicating partner. However, the TLV structure can not be used for speech packet transfers over low bandwidth channels, as the additional header material would consume too much bandwidth for proper operation. For this reason, on the other hand, we designed a second transport mechanism with minimized header material specifically for speech data transfer.

**encryption and message authentication**
We implemented encryption and message authentication with the help of the libtomcrypt [109] library. For encryption we use AES-256 in Counter (CTR) mode as described in Section 3.1, whereas the IV (initialization Vector) is derived from the shared master secret during AKE. For message authentication we use HMAC-SHA256 with truncation to no more than $\frac{L}{2} = 128$ bit as described in Section 3.1.

Initially, if Alice and Bob wish to set up a secure call, our encrypted speech communication application always runs the AKE with SAS- or KCM-based authentication to ensure that there was no MitM attack and the other party really is who she claims to be. There are three cases we deal with:

1. If two parties run AKE for the first time, a SAS is created which is compared out of band. If the comparison was successful, a new entry is made in the "known parties" file so that KCM can be used for future authentications (see Section 4.4 and **KCM** above).

2. Otherwise, due to KCM, both parties already know the long term public keys of each other which the application uses to verify their identities (see Section 4.4 and **KCM** above).

3. A special case occurs if only one party knows the long term public key of the other party. In this case we use SAS for authentication, but in addition the party already having the public key in the "known parties" file verifies that the received public key really matches the stored key.

Once the AKE has completed, both parties have exchanged their long-term public keys and they share vital key material such as the shared master secret and MAC keys. In case two parties connect to each other over a network, chances are high that they already know their own address and the address of the remote party. Yet, there are scenarios where this is not the case which would render KCM authentication impossible. Essentially, there are two possibilities:

1. The caller knows her own address and the address (i.e. the phone number) of the other party (i.e. the callee). As the caller already has all required information, additional steps are not necessary.

2. The callee knows her own address, but she does not know the address of the caller. Therefore, it is required that someone (i.e. the caller or the network stack underneath) tells her who is calling.

For this reason, after performing the AKE, our application *exchanges* the addresses of the communicating partners over the secure channel:

1. If Alice and Bob connect to each other the first time, we first perform AKE. Once AKE has completed successfully, we exchange the encrypted and MAC authenticated addresses of both parties. After that, the SAS is created and compared out of band. In case SAS based authentication was successful, the received address together with the corresponding long-term public key of the other party is stored in the key continuity database. This way also the exchanged addresses are authenticated through SAS.

2. If Alice and Bob already performed AKE earlier, there is an entry with the addresses and the long-term public keys of each other in their KCM files. In this case, just like for case (1) above, we first perform AKE and then securely exchange the addresses of each other. As they both already have an entry for each other in their KCM file, we can now compare the received information (i.e. the remote address and the long-term public key) with the stored information. If it matches, authentication was successful. Otherwise chances are high that there was an attack, a warning message is displayed and the connection is terminated.

3. If only either Alice or Bob have an entry for the remote party in their KCM file, we first perform AKE and then securely exchange their addresses. Yet, on the side of the party not having an entry in the KCM, we send a request that SAS authentication should be used instead of KCM. Therefore, if for instance Alice already knows Bob, but not vice-versa, Bob would request SAS authentication from Alice. Alice will in turn perform SAS authentication with Bob, but in addition she will also check that the information received from Bob matches the entry in the KCM file. This avoids a potential vulnerability in which an attacker could use a fall back to SAS based authentication to mount a potential impersonation attack. (Ultimately he would still have to pass SAS based authentication, but KCM based authentication is stronger in comparison to SAS based authentication if SAS based authentication is not done out-of-band but over the encrypted speech link instead.)

To give an example for SAS based authentication, the caller (e.g. Alice) could see a dialog similar to this one:

```
----------------- SAS authentication (caller) -----------------
Please read these words to your communication partner:
preshrunk hurricane village maverick

Check that the reponse from the other party matches the following words:
talon tambourine snapline Cherokee
---------------------------------------------------------------
```

The callee (e.g. Bob) would now hear the words from Alice and respond in case the words matched:

```
----------------- SAS authentication (callee) -----------------
You should now hear the following words from your communication partner:
preshrunk hurricane village maverick

If these words match, please respond with:
talon tambourine snapline Cherokee
---------------------------------------------------------------
```

We would like to mention that, although adequate, in-band SAS based authentication is the weakest form of authentication that we support. If the SAS is compared out-of-band (e.g. the persons meet and visually compare it), the mechanism is more secure. However, once initial SAS based authentication was successfully performed, the stronger KCM variant is used per default. The overall application design including the cryptographic implementation has roughly 7400 lines of C source code. The synopsis of the application can be seen in Listing 5.3:

**Listing 5.3: Synopsis of the encrypted speech communication application**

```
Usage: ./phone
(-i|--identity)  identity of this program (i.e. our telephone number)
(-d|--device)    serial port device to use
[-a|--address]   address of the party we would like to call
                 (i.e. the phone number of the callee)
[-s|--speed]     serial port baudrate to use (default: 9600)
[-k|--keyprefix] prefix for our ECDSA keypair
                 (i.e. PREFIX.pub, PREFIX.priv), default: "key"
[-t|--authfile]  file used to store authentication information, relative
                 to keypath
[-p|--keypath]   path where the long term ECDSA keys are stored
                 (local directory is the default)
[-w|--password]  password for our private key
[-r|--responder] act as responder
[-e|--echo]      instead of real communication, echo back speech packets
[-h|--help]      print this help
```

The standard packet header to transport encrypted messages is visible in Listing 5.4.

**Listing 5.4: Standard packet header**

```
typedef struct {
 uint8_t packet_type;
 uint16_t payload_length;
 uint32_t sequence_number;
 uint8_t header_crc;
} __attribute__((__packed__)) packet_header_t;
```

The *packet_type* specifies whether the packet is either a speech packet or it contains TLV (Type, Length, Value) information. In case of speech packets the header in Listing 5.5 is used.

**Listing 5.5: Header for speech packets**

```
typedef struct {
 uint8_t bitlen;
} __attribute__((__packed__)) speech_header_t;
```

TLV packets use the header visible in Listing 5.6.

**Listing 5.6: Header for TLV packets**

```
typedef struct {
 uint8_t field_type;
 uint8_t field_len;
} __attribute__((__packed__))field_header_t;
```

CHAPTER 6

# Evaluation and results

In our test setup, we connected two encrypted speech communication units over a null-modem serial cable (Fig. 6.1). By changing the speed of the serial port on both devices we were able to conduct measurements and functionality tests with different bandwidth constraints.

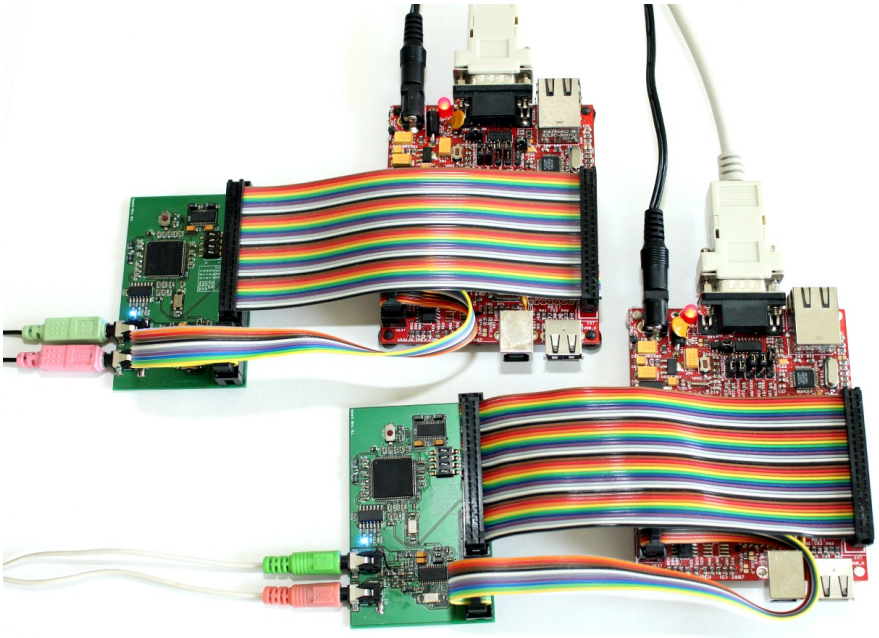

Figure 6.1: test setup: two interconnected encrypted speech communication units

## 6.1 Duration of the key exchange

Our Authenticated Key Exchange (AKE) implementation (see Sections 4.2 and 5.2) is dimensioned for 256-bit cryptographic system security. Hence, message fields that need to be exchanged (such as the EC-DH and EC-DSA public keys, HMACs or signatures) all have a length corresponding to 256-bit system security (in accordance with the requirements issued by standardization bodies such as NIST). Considering the low bandwidth of the system, transferring these fields takes a significant amount of time. In addition, also the amount of heavy cryptographic computations involved with the key exchange are relevant.
During the AKE, in our implementation the call initiator needs to send 376 bytes to the responder, whereas the responder needs to send 345 bytes to the initiator. That is, throughout the key exchange, a total of 721 bytes need to be exchanged. We measured the duration of the key exchange at different baud rates. The results can be seen in Table 6.1. Note that these results also include the constant cryptographic computation delays.

| baud rate | avg. duration at initiator [ms] | avg. duration at responder [ms] |
|---|---|---|
| 4800 | 3855 | 2625 |
| 9600 | 3090 | 2160 |
| 19200 | 2715 | 1930 |

Table 6.1: duration of the AKE at different baud rates

## 6.2 AMBE speech codec latency

In our application the AMBE-3000 [92] codec DSP is connected over a UART interface running at a baudrate of 460.800 kbaud. To compress a speech packet, the packet needs to be transferred to the DSP. As soon as the DSP is ready, the compressed packet is transferred back to the host. However, since we use a pipelined approach to exchange packets with the DSP (see 5.2 for details), the data transfers are done concurrently to the compression and decompression operations. As a result, the data transfers do not have any influence on the total codec delay.
To experimentally measure the codec delay (and thus the additional amount of latency it causes to the system), we modified our implementation to create time stamp logs. Each time we sent an uncompressed speech packet to the DSP, we created a time stamp right before the call to the *write()* system call. Similarly, we created a time stamp right after the *read()* system call. At that point, we had to ensure that the response really is a corresponding, compressed speech packet. By subtracting these time stamps we were able to measure delays. Additional delays (e.g. between the call to *write()* and the physical UART send operation) can be neglected due to the pipelined approach we mentioned above. In contrast to the AMBE-3000 manual [92], which states that the total algorithmic delay (i.e. the total delay due to the compression and decompression) is 62 ms, our measurements showed that compression and decompression each take 10 ms on average. Hence, according to our measurements, the total algorithmic delay was 20 ms. Our tests also showed that the measured algorithmic delay does not depend on the configured AMBE speech

rate. Thus, for instance, the measurement results for lower (e.g. 2400 baud) and higher (e.g. 9600 baud) speech rates were equal.

## 6.3 Speech communication latency and required bandwidth

In our implementation the speech communication latency and required bandwidth are closely tied to each other. The reason for this is that in addition to the speech frames, each encrypted speech packet also includes:

- a "standard" header with 8 bytes (see Listing 5.4)
- a speech header with 1 byte (see Listing 5.5)
- the HMAC for message authentication with at least 16 bytes [65, 111].

Hence, if a speech packet is transferred, it always includes at least $8 + 1 + 16 = 25$ bytes of required header material. The size of a compressed speech frame (i.e. a *channel packet*) is visible in Table 6.2. It depends on the configured AMBE-3000 [92] speech rate, where the *vocoder_rate_index* specifies a set of predefined speech rates [92]. Exactly one compressed speech frame is created each 20 ms.

| vocoder_rate_index | speech_rate | bits_per_channel_packet (=speech_rate/50) | channel_bytes_per_packet (=((bits−1)/8)+1) |
|---:|---:|---:|---:|
| 36 | 2250 | 45 | 7 |
| 37 | 2400 | 48 | 7 |
| 38 | 3000 | 60 | 9 |
| 39 | 3600 | 72 | 10 |
| 40 | 4000 | 80 | 11 |
| 41 | 4400 | 88 | 12 |
| 42 | 4800 | 96 | 13 |
| 43 | 6400 | 128 | 17 |
| 44 | 7200 | 144 | 19 |
| 45 | 8000 | 160 | 21 |
| 46 | 9600 | 192 | 25 |

Table 6.2: different AMBE-3000 speech rates

Since the size of the required header material is usually larger than the size of a compressed speech frame (with the exception of the 9600 bps bit rate), sending only a single speech frame in each packet would tremendously increase the required bandwidth. Therefore, it is much more bandwidth efficient to include a number (*SPEECH_FRAMES_IN_CRYPTO_PACKET*) of speech frames in each packet. But as each speech frame represents 20 ms of speech, this also increases the latency by (*SPEECH_FRAMES_IN_CRYPTO_PACKET**20 *ms*).

We created a model to evaluate the required bandwidth and the resulting latency. In the following, we also verified the model with actual measurement results. If we assume that

| | |
|---|---|
| s ··· | packet size, |
| i ··· | SPEECH_FRAMES_IN_CRYPTO_PACKET, |
| r ··· | vocoder bit rate, |
| L ··· | HMAC size, |
| b ··· | required baud rate, |

the size of one packet is:

$$s = 8 + 1 + i(\frac{\frac{r}{50} - 1}{8}) + L$$
$$s = L + i(\frac{r}{400} - \frac{1}{8}) + 9$$

The speech codec creates one compressed speech frame each 20ms, thus in one second it is required to send $\frac{50}{i}$ packets. If we assume that the 8N1 format is used for serial communication, the required baud rate is:

$$b = \frac{500(i(\frac{r}{400} - \frac{1}{8}) + L + 9)}{i}$$

Therefore, for a given setup (maximum baud rate, bit rate, HMAC size) the minimum number of required speech frames per packet $i$ is:

$$i = \frac{40(b - 90)}{r - 50(400L + 1)}$$

The maximum possible bit rate $r$ for a given system (maximum baud rate, speech frames per packet, HMAC size) is:

$$r = \frac{2(2bi + 125(i - 8(L + 9)))}{5i}$$

Finally, the maximum possible HMAC size $L$ for a given setup (maximum baud rate, speech frames per packet, bit rate) is:

$$L = \frac{-(5ir - 2(2bi + 125(i - 72)))}{2000}$$

Assuming that a 128 bit HMAC ($L = 16$) is used, we determined the possible AMBE bit rate and additional latency tradeoff for 4800 and 9600 baud channels. The results are shown in Fig. 6.2 and Fig. 6.3 respectively. As can be seen, the latency is reducible by choosing a lower vocoder bit rate which allows us to include fewer compressed speech frames in each encrypted packet. However, at the same time this also reduces the speech quality of the system.

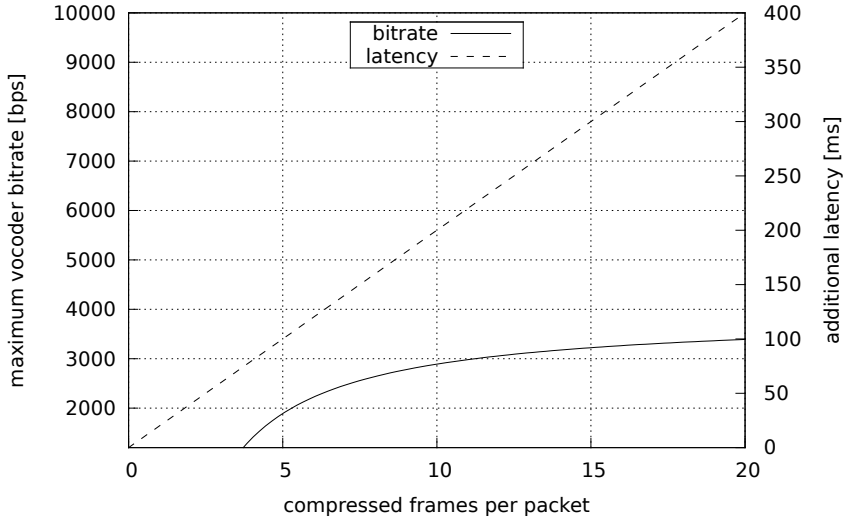


Figure 6.2: possible bit rates at 4800 baud with 128 bit HMAC size

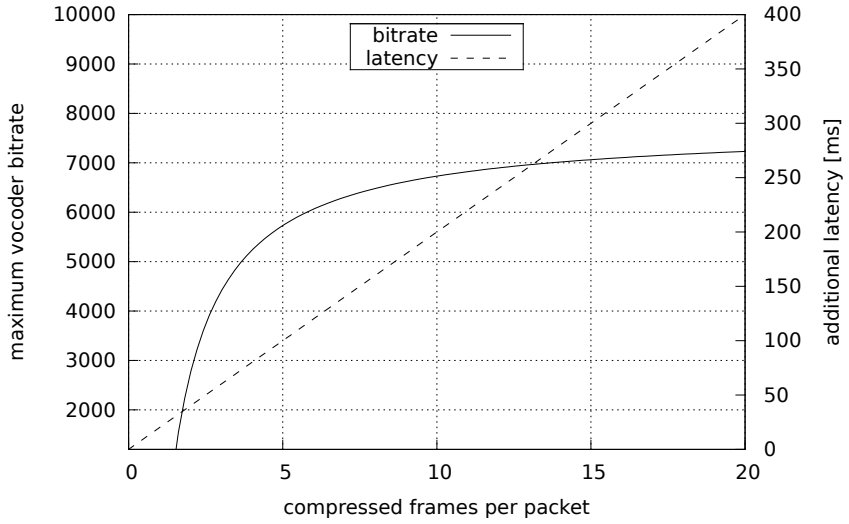


Figure 6.3: possible bit rates at 9600 baud with 128 bit HMAC size

On a $4800$ baud channel, by choosing the minimum vocoder bit rate of $2250$ bps and a $128$ bit HMAC field, we would need to include at least $6$ compressed speech frames in each encrypted packet which would result in $120$ ms of additional latency. In comparison, on a $9600$ baud channel, the same configuration would only require $2$ frames per packet and thus result in an additional latency of merely $40$ ms. The relation between latency, the number of speech frames per packet and the required baudrate is depicted in Fig. 6.4.

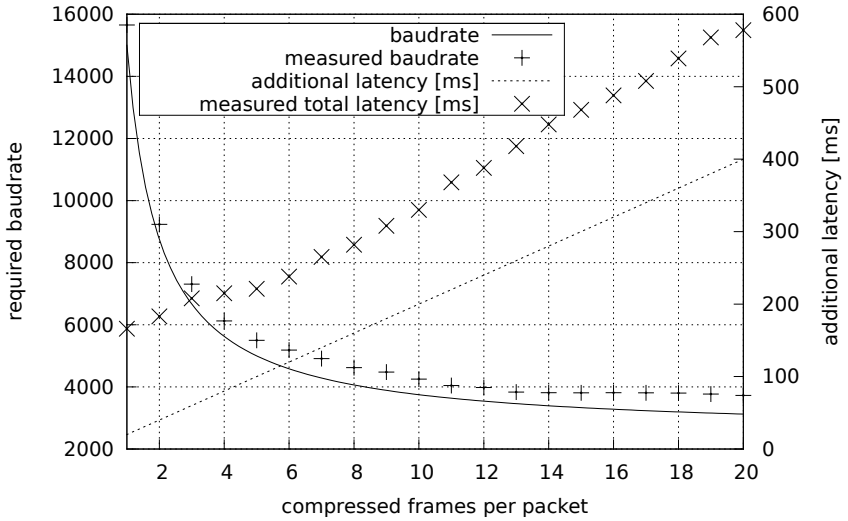


Figure 6.4: required baudrate for the lowest vocoder bit rate ($2250$ bps) and $128$ bit HMAC

Using timestamps as described in Section 6.2, we also measured the required baudrate and the total system latency. The measurement results are depicted in Table 6.3. We incorporated them into Fig. 6.4 as well. While the measured baudrate closely matches the calculated required baudrate, the discrepancy between the calculated *additional* latency and the measured *total* latency comes from the fact, that constant latencies (e.g. due to buffering) are not included in the calculated *additional* latency. In our test setup we used a *playback* buffer with $5$ frames introducing an additional latency of $5 * 20 = 100$ ms. Thus, the total system latency could be further decreased at the cost of smaller buffers and thus less resistance to jitter.

| frames per packet | measured baud rate | measured latency |
|---:|---:|---:|
| 1 | 15655 | 166 |
| 2 | 9235 | 183 |
| 3 | 7310 | 208 |
| 4 | 6125 | 215 |
| 5 | 5500 | 221 |
| 6 | 5185 | 238 |
| 7 | 4913 | 265 |
| 8 | 4623 | 282 |
| 9 | 4476 | 308 |
| 10 | 4250 | 330 |
| 11 | 4044 | 368 |
| 12 | 3980 | 388 |
| 13 | 3832 | 418 |
| 14 | 3815 | 448 |
| 15 | 3810 | 468 |
| 16 | 3815 | 488 |
| 17 | 3810 | 508 |
| 18 | 3800 | 539 |
| 19 | 3770 | 568 |
| 20 | 3725 | 578 |

Table 6.3: measured baud rates and latency

CHAPTER 7

# Related work

While encrypted speech communication is not new, we think that the security features and information security properties we applied to audio streams are novel, especially in conjunction with low-bandwidth communication. Toorani and Beheshti Shirazi [5], for instance, suggested different solutions to the GSM security weaknesses including replacements for insecure ciphering algorithms in mobile networks and end-to-end security solutions. They regarded end-to-end security to be the "easiest, and most profitable solution" [5] and considered different implementations targeting mobile phone hardware or portable PCs.
As we outlined in Chapter 8, GSM has a 9600 kbit/s data channel (CSD - Circuit Switched Data) that can be used in transparent (i.e. *without* error correction) or in non-transparent mode (i.e. with ARQ and FEC error correction) [112]. The drawback of CSD is that the data channel has less bandwidth and does not have the hard real-time properties of the GSM speech channel resulting in higher delays. On the other hand, data access to the raw GSM speech channel is usually not possible for mobile phone users. Instead, only speech can traverse the established channel, which is always processed by the GSM speech codec. This led to a number of end-to-end security solutions, modulating encrypted data streams in the form of a speech-like waveforms that can pass GSM compression without loosing too much information.

In their paper "Real-time End-to-end Secure Voice Communications Over GSM Voice Channel" Katugampala et al. [113] mentioned these drawbacks and designed a modem for low bit rate speech channels. They implemented a prototype on Personal Computers (PCs) and interfaced it to GSM handsets by connecting the PC's sound cards to hands-free cables. Their modem channel achieved a throughput of 3 kbps over which they were able to transfer encrypted speech that was compressed through a LPC based speech codec. As their main contribution was the design of the modem, they provided no information regarding the security features, encryption or key exchange.

In the paper "Secure Mobile Communication Using Low Bit-rate Coding Method" Wastif et al. [114] applied the ITU-T G.723.1 speech coder to compress speech. They encrypted it and

created a speech-like waveform by using a speech production model based on LPC to be able to send the data over the GSM speech channel. Similar to Katugampala et al., they provide no information regarding encryption, key-exchange or security features. Their main contribution is a MATLAB simulation, demonstrating that secure mobile communication over the GSM speech channel is feasible by using speech-like waveforms.

Likewise, Yang et al. [115] and Islam et al. [116] focused on speech-like waveform modem implementations in simulation environments such as MATLAB, without going into details regarding the security properties of their systems.

In the paper "Developing and Implementing Encryption Algorithm for Addressing GSM Security Issues", Islam and Ajmal [117] took a different approach. They implemented a speech encryption system on DSP Starter Kits that uses their own customly developed "embedded encryption algorithm". Besides not being scrutinized by the cryptographic community, the algorithm merely shifts the quantized speech signal from the microphone in a way similar to the antique Caesar cipher. For simplicity, they only considered 16 quantization levels and argued that their encryption algorithm allows "speedy processing and lesser latency". However, their implementation also supports "commonly used encryption algorithms". Apart from their "embedded encryption algorithm", they make no claims regarding the key-exchange or the security properties.

In contrast to the above mentioned implementations, our system neither focuses on speech-like waveforms nor on GSM alone. Instead, it requires a low-bandwidth data communication channel with at least $4800$ baud. We believe our system is superior to those mentioned, as it fully implements authenticated key exchange, key continuity management and encryption relying solely on established cryptographic primitives that are widely-used and scrutinized by the cryptographic community. Moreover, our implementation is a working embedded prototype rather than a simulation (like some of the above mentioned projects).

Lei, Zhao, Dai and Wang [118] proposed a secure voice communication systems based on the TMS320VC5410 DSP, whereas their system also includes a PCM codec and a LCD based user interface. They use uncompressed speech at a sampling rate of $8$ kHz and $8$ bit resolution, which results in a minimum bandwidth requirement of $64$ kbit/s. To encrypt speech data, they designed a "chaotic audio encryption scheme" comprising cat map based diffusion and logistic map based permutation. Their customly designed encryption scheme has not undergone the scrutiny of the cryptographic community and they make no claims regarding its security. Similar to our approach, they tested their solution over a serial E/A-232 connection. Anas et al. [119] implemented a DSP based secure speech communication system utilizing DES for encryption and ICELP for speech compression. Their system uses a modem connection over PSTN and is capable of full-duplex communication starting at $9600$ bps channel bandwidth. In both publications, the authors did not cover the security features or the key exchange of their respective systems.

In [120], Wong and Ching propose a design of a narrowband radio channel speech encryption system. The authors discuss different modulation schemes and speech coding algorithms like CELP, concluding that CELP based vocoders show "considerable promise for good quality coding

of speech at low bit rates" [120]. Likewise, in "Interoperable secure voice communications in tactical systems" Collura and Rahikka from the National Security Agency covered different speech coding techniques, usable for secure voice communication over low-bandwidth channels. In contrast to Wong and Ching, they included Forward Error Correction (FEC) codes in their test setup. Due to the main focus being on speech compression, the authors of both papers did not cover security features or cryptographic properties.

To the best of our knowledge, Hua et al. [121] are the only authors who adopted Advanced Multi-band Excitation (AMBE) for speech coding. They implemented a secure speech communication system that communicates over low-bandwidth IEEE 802.15.4 Wireless Personal Area Networks (WPANs). Their system is based on IEEE 802.15.4 evaluation and AMBE VC-55 vocoder boards. However, rather than implementing encryption on their own, they utilize the security suite in IEEE 802.15.4 that offers AES-128 encryption and integrity protection. They mention that IEEE 802.15.4 has no key management and generation algorithms, but they do not tackle the problem in their implementation.

At the Università degli Studi di Salerno, there is a research project entitled "SPEECH: Secure Personal End-to-End Communication with Handhelds" [122, 123]. According to the publication by A. Castiglione et al. [123], they implemented a SPEECH software application usable to make secure calls on Windows Mobile handheld devices. It uses a 9600 bps communication channel and focuses on the information security properties confidentiality, authentication and non-repudiation. The authors claim, that especially non-repudiation for encrypted speech communication is "absolutely novel" [122] in the scientific community. Their system uses Speex for voice compression and AES-256 in OFB mode for encryption. It allows the use of "SSL/TLS, Passphrase or Diffie-Hellman, for session key agreement and parties authentication" [122]. However, according to [123], these information security properties are not always achieved, as they depend on the type of key-exchange. If merely the *unauthenticated Diffie-Hellman* key-exchange is used, their system neither provides authentication nor non-repudiation. If their *passphrase based key agreement* scheme is applied, their system provides only *weak* authentication, as in practical applications the security of an underlying passphrase is usually not as strong as generated key material with state of the art length. If a user chooses a bad passphrase, this key agreement scheme may lead to a complete compromise of confidentiality and authenticity. The only key agreement scheme, that provides all three security properties at the same time, is the *certificate based* scheme. The drawback of this approach is that all devices need to have the root CA certificate and thus, trust is extended not only to the communicating parties, but to *all parties* having a valid certificate signed by the CA. Due to the lack of online access to a PKI, important security features such as access to CRLs (Certificate Revocation Lists) are not possible. If a SPEECH device is stolen, it still contains valid certificates signed by the CA, but due to the lack of CRLs, there is no way to revoke the trust from these stolen certificates. Besides, SPEECH implemented a key escrow scheme allowing third parties (e.g. law enforcement) to intercept encrypted conversations without notice. In comparison, our implementation provides a higher degree of security. In addition to the security properties confidentiality and authenticity, it provides integrity protection, perfect forward secrecy and repudiation, whereas, unlike SPEECH, our system *always* grants these properties. We did not implement key escrow schemes as we strongly believe that they are a threat

to communication security. Moreover, the SPEECH key exchange can take up to 12 seconds on a 9600 bps channel, whereas our key exchange takes roughly 3 seconds on average, even on a channel with as little as 4800 baud.

Finally, the work of Goldberg et al. on OTR [77, 1, 81, 82], which forms the basis of our own key exchange, is noteworthy. Interested readers are forwarded to Chapter 4 for more details on the protocol.

In contrast to the above implementations, our system does not only provide **confidentiality**, **integrity** and **authenticity (of data *and* entities)** with well-established and standardized algorithms, but also provides **non-repudiation**, **perfect forward secrecy** and a degree of **plausible deniability** *at the same time*. We believe that these information security features have never been applied to audio streams before and thus are a novel contribution to the field. Our system is a fully working embedded prototype implementation rather than a simulation or an application that only works on specific devices (like Windows Mobile handheld devices). Unlike mentioned projects, our solution is extremely versatile through the use of Linux and general purpose ports such as USB. It does not restrict its use to specific media like GSM, but similar to SPEECH, it has a generic design that allows its use over a number of varying media. Finally, our system also works over channels with bandwidth capacities below 9600 baud. Our tests showed, that the expected lower bound of 4800 baud still provides enough bandwidth for the system to remain operational.

CHAPTER 8

# Conclusion and future work

Throughout a time of roughly 2 years, we developed a working prototype of a versatile encrypted speech communication system. Due to the numerous configuration parameters the system can be easily adapted to work over a broad range of varying communication media. Among these parameters are:

- variable buffer sizes
- variable speech encoding bit rate
- variable tradeoff between latency and required bandwidth

By adapting the buffer sizes, the system can be made more resilient to jitter at the cost of additional latency. The speech encoding bit rate allows to increase the audio quality at the cost of increased bandwidth requirements. Our system allows to choose the tradeoff between latency and required bandwidth by specifying how many speech frames should be included in each encrypted packet. The more frames are included, the lower the required bandwidth at the cost of additional latency (see Chapter 6 for details).
By utilizing the diversity of the Linux operating system, our system enables usage of plug and play devices such as USB headsets or even Bluetooth devices. Our software implementation comprises more than 16.000 lines of C source code specifically designed in a generic and Linux-compatible way, so that the components of our system can be applied by other applications as well. For instance, our sound hardware can be used by arbitrary sound applications, allowing applications like MP3 playback, arbitrary sound recording and many more. Therefore, the speech compression plugin is usable by arbitrary sound applications. For example, it could be used to record voice and play it back at a later point in time. Likewise, it could be used as AMBE de-/compression device in conjunction with other systems that use the AMBE speech codec (such as the D-STAR amateur radio digital voice communications network [124] or the APCO Project 25 trunked radio system [125]).

However, our current implementation also has limitations. In the measurements we performed with our specific setup, the overall communication latency was in the 200 ms range which is more than the 150 ms limit suggested by the G.114 ITU recommendation [126]. While practical experimentation has shown that the latency is acceptable, very low bandwidth constraints further increase the system latency. For instance, on a 4800 baud channel, our measurements (see Table 6.3) have shown that even at the lowest bit rate the latency reaches the 300 ms limit.
Also, in contrast to the serial cable connection in our test setup, a real world communication medium like GSM or a digital radio would introduce jitter and bit error effects we did not test. One way to tackle **jitter** issues is to increase the buffer sizes which will however increase the communication latency. It is up to future work to perform according tests by introducing various amounts of jitter.
On the serial cable connection the **Bit Error Rate (BER)** is practically 0%. Nevertheless, on a real word channel the BER is ordinarily higher. To deal with bit errors a number of error-control mechanisms can be used. For instance, a noisy communication channel with a given number of bit errors could implement an Automatic Repeat reQuest (ARQ) scheme, so that each time a faulty frame is received, the receiver automatically requests the frame to be re-sent. Obviously such a scheme would need an error detection mechanism like Cyclic Redundancy Check (CRC) codes as well, so that the system can detect whether a received frame contains bit errors. Since the frame requesting the sender to re-send the last frame, or the re-sent frame itself, could contain bit errors as well, depending on the BER, it might be relevant to add error-correction mechanisms like Forward Error Correction (FEC) to the system. That is, a certain percentage of the channel bandwidth dependent on the expected BER would be used for redundant error correction codes. In combination with an ARQ scheme, errors could be corrected as long as the number of bit errors is below the threshold, the error correcting code can handle. If the number of errors is too high, the frame can be re-requested with the ARQ scheme. The combination of an ARQ scheme with FEC codes is commonly known as Hybrid Automatic Repeat Request (HARQ) which is part of our future work. Obviously both mechanisms would have an impact on our speech communication system. Re-requesting a frame would introduce additional delays that might break the real-time property of the system. Hence it could occur that due to ARQ speech frames arriving so late, they would be no longer relevant. Instead of correctly receiving those frames, it would we preferable to drop them in favor of keeping the real-time property of the speech data stream. On the other side, control frames (e.g. for call setup and teardown) need to correctly arrive eventually. Ultimately, this leads to different requirements for speech and control packets. Speech frames need to arrive *on time* and dropping erroneous frames is acceptable. On the other hand control frames are not time critical, but they need to arrive correctly eventually.
GSM, for instance, allows to transfer data over the Circuit Switched Data (CSD) channel either in transparent or non-transparent mode [112]. In transparent mode, the raw frames from the Over The Air (OTA) interface are directly passed to the CSD channel. Thus, these frames also contain transmission errors that need to be corrected and, in our current implementation, these errors would cause our system to discard the packets and fail. On the other hand, in non-transparent mode, there are no transmission errors as GSM corrects them by means of retransmission and error correction codes. In this case our system would work over the channel, but due to the re-transmission delays our implementation would drop speech packets as they do no longer arrive

on time and fail as well. Ultimately, to tackle the problem, we would need to implement Hybrid Automatic Repeat Request (HARQ) and communicate over the raw transparent channel. This way we can limit re-transmissions to control packets to guarantee that they will arrive eventually, while we can drop speech packets that can not be corrected through Forward Error Correction (FEC) alone.
Finally, we would like to point out, that our implementation is a prototype. Hence the size of the hardware implementation is still too big to be easily portable, there is no user interface and in the current state the system is not battery powered and thus incapable for mobile use. As more powerful ARM controllers are readily available today (such as the ones being heavily used in smart phones), it would be possible to implement a much smaller and more powerful system. Also recent advances in speech codec development (like the open source Codec2 [98]) would allow us to perform the speech compression in software and omit the requirement for an external DSP. For these reasons, we think that the system could be miniaturized to the size of a match box. Likewise, if existing communication devices (such as smart phones) already have a powerful enough CPU as well as audio capture and playback capabilities, the system could be implemented on the communication device itself making additional hardware obsolete.

# Bibliography


[1] Chris Alexander and Ian Goldberg, "Improved user authentication in off-the-record messaging", in *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, New York, NY, USA, 2007, WPES '07, pp. 41–47, ACM.

[2] Atmel Corporation, "AT91 ARM Thumb Microcontrollers, AT91SAM9260", `http://www.atmel.com/dyn/resources/prod_documents/doc6221.pdf`, [Online; retrieved 2011-05-05].

[3] OLIMEX Ltd., "SAM9-L9260 DEVELOPMENT BOARD FOR AT91SAM9260 MICRO-CONTROLLER", `http://www.olimex.com/dev/sam9-L9260.html`, [Online; retrieved 2011-05-05].

[4] Ross J. Anderson, *Security engineering - a guide to building dependable distributed systems (2. ed.)*, Wiley, 2008.

[5] Mohsen Toorani and Ali Asghar Beheshti Shirazi, "Solutions to the GSM Security Weaknesses", *CoRR*, vol. abs/1002.3175, 2010.

[6] Orr Dunkelman, Nathan Keller, and Adi Shamir, "A Practical-Time Attack on the A5/3 Cryptosystem Used in Third Generation GSM Telephony", `http://eprint.iacr.org/`, 2010, [Online; retrieved 2011-05-05].

[7] Official Journal C 329, "European Council Resolution of 17 January 1995 on the Lawful Interception of Telecommunications", `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31996G1104:EN:HTML`, November 1996, [Online; retrieved 2011-05-05].

[8] Wikipedia, "Greek telephone tapping case 2004-2005", `http://en.wikipedia.org/wiki/Greek_telephone_tapping_case_2004-2005`, 2010, [Online; retrieved 2011-05-05].

[9] "GSMK Gesellschaft für sichere Mobile Kommunikation mbH", `http://www.cryptophone.de`, [Online; retrieved 2011-05-05].

[10] "Redphone 0.1", `http://whispersys.com`, [Online; retrieved 2011-05-05].

[11] "Secure Phone Miser", http://www.spygadgets.com/secure-phone, [Online; retrieved 2011-05-05].

[12] Gesellschaft für sichere Mobile Kommunikation mbH, "Cryptophone Technology - Encryption Engine", http://www.cryptophone.de/en/background/cryptophone-technology/encryption-engine, [Online; retrieved 2011-05-05].

[13] J. Callas P. Zimmermann, A. Johnston, "ZRTP: Media Path Key Agreement for Unicast Secure RTP; Internet-Draft", http://tools.ietf.org/html/draft-zimmermann-avt-zrtp-22, [Online; retrieved 2011-05-05].

[14] "The ZFone Project", http://zfoneproject.com, [Online; retrieved 2011-05-05].

[15] "Off-the-Record Messaging Protocol version 2", http://www.cypherpunks.ca/otr/Protocol-v2-3.1.0.html, [Online; retrieved 2011-05-05].

[16] Dirk Pesch, "Lecture Notes on Telecommunications", http://www.aws.cit.ie/personnel/dpesch/MScSW_Telecomms.html, [Online; retrieved 2011-05-05].

[17] Jörg Eberspächer, Hans-Jörg Vögel, Christian Bettstetter, and Christian Hartmann, *GSM - Architecture, Protocols and Services*, Wiley, 3rd edition edition, December 2008.

[18] ETSI, "ETSI TS 123 003 V9.3.0 (2010-06) Technical Specification; Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); Numbering, addressing and identification; (3GPP TS 23.003 version 9.3.0 Release 9), 3GPP TS 23.003", Nov. 2005.

[19] Oracle Corporation, "Java Card Technology Overview", http://java.sun.com/javacard/overview.jsp, [Online; retrieved 2011-05-05].

[20] Suraj Srinivas, "The GSM Standard (An overview of its security)", http://www.sans.org/reading_room/whitepapers/telephone/gsm-standard-an-overview-security_317, [Online; retrieved 2011-05-05].

[21] Billy Brumley, "A3/A8 & COMP128, T-79.514 Special Course on Cryptology", http://www.tcs.hut.fi/Studies/T-79.514/slides/S5.Brumley-comp128.pdf, [Online; retrieved 2011-05-05].

[22] Eric Zenner, "Kryptographische Protokolle im GSM Standard: Beschreibung und Kryptanalyse", Master's thesis, Universität Mannheim, 1999.

[23] Wikipedia, "A5/1", https://secure.wikimedia.org/wikipedia/en/wiki/A5/1, 2010, [Online; retrieved 2011-05-05].

[24] Andrey Bogdanov, Thomas Eisenbarth, and Andy Rupp, "A Hardware-Assisted Realtime Attack on A5/2 Without Precomputations", in *Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, Berlin, Heidelberg, 2007, CHES '07, pp. 394–412, Springer-Verlag.

[25] Elad Barkan, Eli Biham, and Nathan Keller, "Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication", *J. Cryptol.*, vol. 21, pp. 392–429, March 2008.

[26] Vassilis Prevelakis and Diomidis Spinellis, "The Athens affair", *IEEE Spectrum*, vol. 44, no. 7, pp. 26–33, July 2007.

[27] Michael Walker, "On the Security of 3GPP Networks", in *EUROCRYPT*, 2000, pp. 102–103.

[28] Bundesamt für Sicherheit in der Informationstechnik, "Öffentliche Mobilfunknetze und ihre Sicherheitsaspekte", `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Broschueren/OeffentlMobilfunk/oefmobil_pdf.pdf?__blob=publicationFile`, [Online; retrieved 2011-05-05].

[29] Holger Bertsch, "Open Source GSM BTS Setup und Analyse für Demo-Zwecke", Master's thesis, Albert-Ludwigs-Universität Freiburg, 2009.

[30] Dennis Wehrle, "Open Source IMSI-Catcher", Master's thesis, Albert-Ludwigs-Universität Freiburg, 2009.

[31] Fabian van den Broek, "Catching and Understanding GSM-Signals", Master's thesis, Radboud University Nijmegen, 2010.

[32] Ettus Research LLC, "Universal Software Radio Peripheral", `http://www.ettus.com/products`, [Online; retrieved 2011-05-05].

[33] "GNU Radio Project", `http://gnuradio.org`, [Online; retrieved 2011-05-05].

[34] "AirProbe project", `https://svn.berlin.ccc.de/projects/airprobe/wiki`, [Online; retrieved 2011-05-05].

[35] "OpenBTS project", `http://openbts.sourceforge.net`, [Online; retrieved 2011-05-05].

[36] Timo Gendrullis, Martin Novotný, and Andy Rupp, "A Real-World Attack Breaking A5/1 within Hours", in *Proceeding sof the 10th international workshop on Cryptographic Hardware and Embedded Systems*, Berlin, Heidelberg, 2008, CHES '08, pp. 266–282, Springer-Verlag.

[37] Karsten Nohl, "Attacking phone privacy, BlackHat 2010 Lecture Notes", `http://media.blackhat.com/bh-us-10/whitepapers/Nohl/BlackHat-USA-2010-Nohl-Attacking.Phone.Privacy-wp.pdf`, [Online; retrieved 2011-05-05].

[38] Dan Boneh, Ed., *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, vol. 2729 of *Lecture Notes in Computer Science*. Springer, 2003.

[39] Security Research Labs, "Decrypting GSM phone calls", http://srlabs.de/research/decrypting_gsm, [Online; retrieved 2011-05-05].

[40] Man Young Rhee, *Mobile Communication Systems and Security*, Wiley Publishing, 2009.

[41] P. Neittaanmäki, T. Rossi, S. Korotov, E. Oñate, J. Périaux, D. Knörzer (eds, Kaisa Nyberg, and Kaisa Nyberg, "Cryptographic Algorithms for UMTS", http://www.tcs.hut.fi/Publications/knyberg/eccomas.pdf, 2004, [Online; retrieved 2011-05-05].

[42] Klaus Vedder, "The UICC - The Security Platform for Value Added Services", http://docbox.etsi.org/Workshop/2009/200901_SECURITYWORKSHOP/G&D_Vedder_UICC_SecurityPlatformforValueAddedServices.pdf, January 2009, [Online; retrieved 2011-05-05].

[43] ETSI, "ETSI TS 135 205 V9.0.0 (2010-02) Technical Specification; Universal Mobile Telecommunications System (UMTS); Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 1: General; (3GPP TS 35.205 version 9.0.0 Release 9)", Feb. 2010.

[44] Noureddine Boudriga, *Security of Mobile Communications*, Auerbach Publications, Boston, MA, USA, 2009.

[45] "ETSI TS 135 206 V9.0.0 (2010-02) Technical Specification; Universal Mobile Telecommunications System (UMTS); Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm specification; (3GPP TS 35.206 version 9.0.0 Release 9), author = ETSI, organization = European Telecommunications Standards Institute", Feb. 2010.

[46] Valtteri Niemi and Kaisa Nyberg, *UMTS security*, Wiley, 2003.

[47] ETSI, "ETSI TS 133 102 V9.3.0 (2010-10) Technical Specification; Universal Mobile Telecommunications System (UMTS); 3G security; Security architecture; (3GPP TS 33.102 version 9.3.0 Release 9)", Oct. 2010.

[48] ETSI, "ETSI TS 135 202 V9.0.0 (2010-02) Technical Specification; Universal Mobile Telecommunications System (UMTS); Specification of the 3GPP confidentiality and integrity algorithms; Document 2: Kasumi specification; (3GPP TS 35.202 version 9.0.0 Release 9)", Feb. 2010.

[49] Blandine Debraize and Irene Marquez Corbella, "Fault Analysis of the Stream Cipher Snow 3G", in *Proceedings of the 2009 Workshop on Fault Diagnosis and Tolerance in Cryptography*, Washington, DC, USA, 2009, FDTC '09, pp. 103–110, IEEE Computer Society.

[50] GSM Association, "GSM Security Algorithms", http://gsmworld.com/our-work/programmes-and-initiatives/fraud-and-security/gsm_security_algorithms.htm, [Online; retrieved 2011-05-05].

[51] Muzammil Khan, Attiq Ahmed, and Ahmad Raza Cheema, "Vulnerabilities of UMTS Access Domain Security Architecture", in *Proceedings of the 2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, Washington, DC, USA, 2008, pp. 350–355, IEEE Computer Society.

[52] Orr Dunkelman, Nathan Keller, and Adi Shamir, "A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3G telephony", in *Proceedings of the 30th annual conference on Advances in cryptology*, Berlin, Heidelberg, 2010, CRYPTO'10, pp. 393–410, Springer-Verlag.

[53] Ulrike Meyer and Susanne Wetzel, "A man-in-the-middle attack on UMTS", in *Proceedings of the 3rd ACM workshop on Wireless security*, New York, NY, USA, 2004, WiSe '04, pp. 90–97, ACM.

[54] Ulrike Meyer and Susanne Wetzel, "On the impact of GSM encryption and man-in-the-middle attacks on the security of interoperating GSM/UMTS networks", in *PIMRC*. 2004, pp. 2876–2883, IEEE.

[55] Zahra Ahmadian, Somayeh Salimi, and Ahmad Salahi, "New attacks on UMTS network access", in *Proceedings of the 2009 conference on Wireless Telecommunications Symposium*, Piscataway, NJ, USA, 2009, WTS'09, pp. 291–296, IEEE Press.

[56] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot, *Handbook of Applied Cryptography*, CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[57] European Network of Excellence in Cryptology II, "ECRYPT II Yearly Report on Algorithms and Key Lengths (2010); Revision 1.0", http://www.ecrypt.eu.org/documents/D.SPA.13.pdf, March 2010, [Online; retrieved 2011-05-05].

[58] Tata Elxsi Limited, "Elliptic Curve Cryptography - An Implementation Guide", http://www.tataelxsi.com/whitepapers/ECC_Tut_v1_0.pdf?pdf_id=public_key_TEL.pdf, [Online; retrieved 2011-05-05].

[59] Tom St Denis Dana Neustadter, "Elliptic Curves over Prime and Binary in Cryptography", http://www.ellipticsemi.com/pdf/presentations/EC_overGF_in_cryptography.pdf, [Online; retrieved 2011-05-05].

[60] Certicom Research, "STANDARDS FOR EFFICIENT CRYPTOGRAPHY, SEC 1: Elliptic Curve Cryptography; Standards for Efficient Cryptography Group (SECG)", http://www.secg.org/download/aid-385/sec1_final.pdf, [Online; retrieved 2011-05-05].

[61] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone, *Guide to Elliptic Curve Cryptography*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[62] Certicom Research, "STANDARDS FOR EFFICIENT CRYPTOGRAPHY, SEC 2: Recommended Elliptic Curve Domain Parameters; Standards for Efficient Cryptography Group (SECG)", http://www.secg.org/download/aid-386/sec2_final.pdf, [Online; retrieved 2011-05-05].

[63] NIST, "RECOMMENDED ELLIPTIC CURVES FOR FEDERAL GOVERNMENT USE", http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf, July 1999, [Online; retrieved 2011-05-05].

[64] National Security Agency, "NSA Suite B Cryptography", http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml, [Online; retrieved 2011-05-05].

[65] NIST, "Federal Information Processing Standards Publication 198; The Keyed-Hash Message Authentication Code (HMAC)", http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf, Mar. 2002, [Online; retrieved 2011-05-05].

[66] R. Canetti H. Krawczyk, M. Bellare, "HMAC: Keyed-Hashing for Message Authentication; Request for Comments: 2104", http://www.ietf.org/rfc/rfc2104.txt, [Online; retrieved 2011-05-05].

[67] NIST, "Federal Information Processing Standards Publication 197; Announcing the ADVANCED ENCRYPTION STANDARD (AES)", http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, Nov. 2001, [Online; retrieved 2011-05-05].

[68] R. Housley, "Using Advanced Encryption Standard (AES) Counter Mode; With IPsec Encapsulating Security Payload (ESP); Request for Comments: 3686", http://www.ietf.org/rfc/rfc3686.txt, [Online; retrieved 2011-05-05].

[69] T. Kohno M. Bellare, "The Secure Shell (SSH) Transport Layer Encryption Modes; Request for Comments: 4344", http://www.ietf.org/rfc/rfc4344.txt, [Online; retrieved 2011-05-05].

[70] C. Lonvick T. Ylonen, "The Secure Shell (SSH) Transport Layer Protocol; Request for Comments: 4253", http://www.ietf.org/rfc/rfc4253.txt, [Online; retrieved 2011-05-05].

[71] C. Lonvick T. Ylonen, "The Secure Shell (SSH) Protocol Architecture; Request for Comments: 4251", http://www.ietf.org/rfc/rfc4251.txt, [Online; retrieved 2011-05-05].

[72] Peter Gutmann, "Underappreciated Security Mechanisms", http://www.cs.auckland.ac.nz/~pgut001/pubs/underappreciated.pdf, [Online; retrieved 2011-05-05].

[73] R. Bresciani and A. Butterfield, "A formal security proof for the ZRTP Protocol", in *Internet Technology and Secured Transactions, 2009. ICITST 2009. International Conference for*, nov. 2009, pp. 1 –6.

[74] Jeremy Robin and Andrew Schwartz, "Analysis of ZRTP", `http://www.stanford.edu/class/cs259/WWW06/projects/project05/05-Writeup.pdf`, [Online; retrieved 2011-05-05].

[75] "On the Security of Short Authentication Strings; post to ietf-rtpsec mailing list, Sat, 17 Mar 2007", `http://www.imc.org/ietf-rtpsec/mail-archive/msg00608.html`, [Online; retrieved 2011-05-05].

[76] Helmut Hlavacs, Wilfried Gansterer, M. Petraschek, Thomas Höher, and O. Jung, "Security and Usability Aspects of Man-in-the-Middle Attacks on ZRTP", *J. Univers. Comput. Sci.*, vol. 14, no. 5, pp. 673–692, 2008.

[77] Nikita Borisov, Ian Goldberg, and Eric Brewer, "Off-the-record communication, or, why not to use PGP", in *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, New York, NY, USA, 2004, WPES '04, pp. 77–84, ACM.

[78] Hugo Krawczyk, "SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols", in *CRYPTO*, 2003, pp. 400–425.

[79] "The Secure Real-time Transport Protocol (SRTP); Request for Comments: 3711", `http://www.ietf.org/rfc/rfc3711.txt`, [Online; retrieved 2011-05-05].

[80] "libsrtp: a library for secure SRTP; About Secure RTP", `http://srtp.sourceforge.net/spec.html`, [Online; retrieved 2011-05-05].

[81] Joseph Bonneau and Andrew Morrison, "Finite-State Security Analysis of OTR Version 2", `http://www.jbonneau.com/OTR_analysis.pdf`, [Online; retrieved 2011-05-05].

[82] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk, "Secure off-the-record messaging", in *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, New York, NY, USA, 2005, WPES '05, pp. 81–89, ACM.

[83] D. Harkins and ed. D. Carrel, "The Internet Key Exchange (IKE); Request for Comments: 2409", `http://www.ietf.org/rfc/rfc2409.txt`, [Online; retrieved 2011-05-05].

[84] C. Kaufman, "Internet Key Exchange (IKEv2) Protocol; Request for Comments: 4309", `http://www.ietf.org/rfc/rfc4309.txt`, [Online; retrieved 2011-05-05].

[85] Ran Canetti and Hugo Krawczyk, "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels", in *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, London, UK, 2001, EUROCRYPT '01, pp. 453–474, Springer-Verlag.

[86] E. Rescorla, "Diffie-Hellman Key Agreement Method; Request for Comments: 2631", http://www.ietf.org/rfc/rfc2631.txt, [Online; retrieved 2011-05-05].

[87] P. Gutmann, "Key Management through Key Continuity (KCM); Internet-Draft", http://tools.ietf.org/id/draft-gutmann-keycont-01.txt, [Online; retrieved 2011-05-05].

[88] Jeff Rodman, "THE EFFECT OF BANDWIDTH ON SPEECH INTELLIGIBILITY", http://www.polycom.com/global/documents/whitepapers/effect_of_bandwidth_on_speech_intelligibility_1.pdf, [Online; retrieved 2011-05-05].

[89] Texas Instruments, "TLV320AIC23B, Stereo Audio CODEC; 8- to 96-kHz, With Integrated Headphone Amplifier", http://focus.ti.com/lit/ds/symlink/tlv320aic23b.pdf, [Online; retrieved 2011-05-05].

[90] Wolfson microelectronics, "WM8731 / WM8731L; Portable Internet Audio CODEC with Headphone Driver and Programmable Sample Rates", http://www.wolfsonmicro.com/documents/uploads/data_sheets/en/WM8731.pdf, [Online; retrieved 2011-05-05].

[91] Jacob Benesty, M. Mohan Sondhi, and Yiteng (Arden) Huang, *Springer Handbook of Speech Processing*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[92] Inc. Digital Voice Systems, "AMBE-3000 Vocoder Chip; Users Manual, Version 1.11", http://www.dvsinc.com/manuals/AMBE-3000_manual.pdf, Nov. 2009, [Online; retrieved 2011-05-05].

[93] S. F. Campos Neto, F. L. Corcoran, J. Phipps, and S. Dimolitsas, "Performance assessment of 4.8 kbit/s AMBE coding under aeronautical environmental conditions", in *Proceedings of the Acoustics, Speech, and Signal Processing, 1996. on Conference Proceedings., 1996 IEEE International Conference - Volume 01*, Washington, DC, USA, 1996, ICASSP '96, pp. 499–502, IEEE Computer Society.

[94] S.-W. Wong, "An evaluation of 6.4 kbit/s speech codecs for Inmarsat-M system", in *Proceedings of the Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference*, Washington, DC, USA, 1991, ICASSP '91, pp. 629–632, IEEE Computer Society.

[95] S. Dimolitsas et.al., "Evaluation of Voice Codec Performance for the Inmarsat Mini-M System", 1995.

[96] "Mobile Services in Australia", IEEE Communications Magazine, Nov. 1991.

[97] Inc. Digital Voice Systems, "DVSI Vocoder Independent Evaluation Results", http://www.dvsinc.com/papers/eval_results.htm, [Online; retrieved 2011-05-05].

[98] "Codec 2 project", http://codec2.org, [Online; retrieved 2011-05-05].

[99] "SoftwareTools - Linux4SAM", http://www.at91.com/linux4sam/bin/view/Linux4SAM/SoftwareTools, [Online; retrieved 2011-05-05].

[100] "AT91Bootstrap - Linux4SAM", http://www.at91.com/linux4sam/bin/view/Linux4SAM/AT91Bootstrap, [Online; retrieved 2011-05-05].

[101] Atmel Corporation, "AT91Bootstrap framework; Version: V1.0, Release Date: 09-Oct-2006", http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4093, [Online; retrieved 2011-05-05].

[102] "Buildroot: making Embedded Linux easy", http://buildroot.uclibc.org, [Online; retrieved 2011-05-05].

[103] "$\mu$Clibc - A C library for embedded Linux", http://www.uclibc.org, [Online; retrieved 2011-05-05].

[104] "Das U-Boot – the Universal Boot Loader", http://www.denx.de/wiki/U-Boot, [Online; retrieved 2011-05-05].

[105] Karim Yaghmour, Jonathan Masters, and Gilad Ben, *Building embedded linux systems, 2nd edition*, O'Reilly & Associates, Inc., Sebastopol, CA, USA, second edition, 2008.

[106] "AT91 Linux 2.6 Patches", http://maxim.org.za/at91_26.html, [Online; retrieved 2011-05-05].

[107] "Linux4SAM Experimental Patches", http://www.at91.com/linux4sam/bin/view/Linux4SAM/LinuxKernel#Linux4SAM_Experimental_Patches, [Online; retrieved 2011-05-05].

[108] "ALSA SoC Layer kernel documentation; *linux-2.6.28/Documentation/sound/alsa/soc*", http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.28.tar.bz2, [Online; retrieved 2011-05-05].

[109] "LibTom Projects - LibTomCrypt", http://libtom.org/?page=features&whatfile=crypt, [Online; retrieved 2011-05-05].

[110] "PGP word list", http://philzimmermann.com/docs/PGP_word_list.pdf, [Online; retrieved 2011-05-05].

[111] NIST, "NIST Special Publication 800-107; Recommendation for Applications Using Approved Hash Algorithms", http://csrc.nist.gov/publications/nistpubs/800-107/NIST-SP-800-107.pdf, [Online; retrieved 2011-05-05].

[112] Agilent Technologies, "GSM Circuit Switched Data (CSD)", http://wireless.agilent.com/rfcomms/refdocs/gsm/gprsla_gen_bse_gsm_csd.html, [Online; retrieved 2011-05-05].

[113] N.N. Katugampala, K.T. Al-Naimi, S. Villette, and A.M. Kondoz, "Real-time End-to-end Secure Voice Communications Over GSM Voice Channel", 13th European Signal Processing Conference (EUSIPCO'05), Turkey, Sept. 2005.

[114] M. Wasif, C.R. Sanghavi, and M. Elahi, "Secure Mobile Communication Using Low Bit-Rate Coding Method", in *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*, sept. 2007, pp. 1410 –1413.

[115] Yucun Yang, Suili Feng, Wu Ye, and Xinsheng Ji, "A Transmission Scheme for Encrypted Speech over GSM Network", in *Proceedings of the 2008 International Symposium on Computer Science and Computational Technology - Volume 02*, Washington, DC, USA, 2008, pp. 805–808, IEEE Computer Society.

[116] S. Islam, F. Ajmal, S. Ali, J. Zahid, and A. Rashdi, "Secure end-to-end communication over GSM and PSTN networks", in *Electro/Information Technology, 2009. eit '09. IEEE International Conference on*, june 2009, pp. 323 –326.

[117] S. Islam and F. Ajmal, "Developing and implementing encryption algorithm for addressing GSM security issues", in *Emerging Technologies, 2009. ICET 2009. International Conference on*, oct. 2009, pp. 358 –361.

[118] Hongyu Lei, Yu Zhao, Yuewei Dai, and Zhiquan Wang, "A secure voice communication system based on DSP", in *Control, Automation, Robotics and Vision Conference, 2004. ICARCV 2004 8th*, dec. 2004, vol. 1, pp. 132 – 137 Vol. 1.

[119] N.M. Anas, Z. Rahman, A. Shafii, M.N.A. Rahman, and Z.A.M. Amin, "Secure speech communication over public switched telephone network", in *Applied Electromagnetics, 2005. APACE 2005. Asia-Pacific Conference on*, dec. 2005, p. 4 pp.

[120] C.K. Wong and P.C. Ching, "Digital speech transmission for highly encrypted and paramilitary operated land mobile radio communications over a narrowband UHF channel", in *Telecommunications, 1991. Third IEE Conference on*, mar 1991, pp. 47 –52.

[121] Lee Yong Hua and Fam Fook Teng, "Delivering high quality, secure speech communication through low data rate 802.15.4 WPAN", in *Telecommunications and Malaysia International Conference on Communications, 2007. ICT-MICC 2007. IEEE International Conference on*, may 2007, pp. 758 –763.

[122] "Secure Personal End-to-End Communication with Handheld", `http://www.speech.dia.unisa.it/en/index.htm`, [Online; retrieved 2011-05-05].

[123] Aniello Castiglione, Giuseppe Cattaneo, Alfredo De Santis, Fabio Petagna, and Umberto Ferraro Petrillo, "SPEECH: Secure Personal End-to-End Communication with Handheld", in *ISSE*, Sachar Paulus, Norbert Pohlmann, and Helmut Reimer, Eds. 2006, pp. 287–297, Vieweg.

[124] "D-Star Info", `http://www.dstarinfo.com`, [Online; retrieved 2011-05-05].

[125] "Project 25 Technology Interest Group", `http://www.project25.org`, [Online; retrieved 2011-05-05].

[126] "ITU-T G.114", `http://eu.sabotage.org/www/ITU/G/G0114e.pdf`, [Online; retrieved 2011-05-05].