Technical status Zifra card

Torbjörn

June 4, 2018

Contents

Abstract

This document is meant to clarify what has been done and what is needed to be done.

1 Brief description

We will here shortly describe what the Zifra card is meant to be. It is an memory card that is supposed to protect data by encrypting the data that is written to it. This is done with the help of multiple key(s) and cipher.

2 System overview

We will here describe the way we want the system to work. For the sake of explanation we will use a camera as an example case. We will first talk about the key generation, secondly the encryption and finally about decryption step.

2.1 Key generation

There are three type of keys used in the crypto scheme. The first two keys are pre-generated as an key pair on a safe device one private key (Pri_key) and one Public key (Pub_key). The pri_key is saved for later use and the pub_key is transferred and saved on the SD card see Figure 1. Then the user puts the SD card in a camera. When the camera boots up a random number is generated this is the Random generated session Key (Rand_key) this number is then encrypted with the Pub_key and saved on the memory.

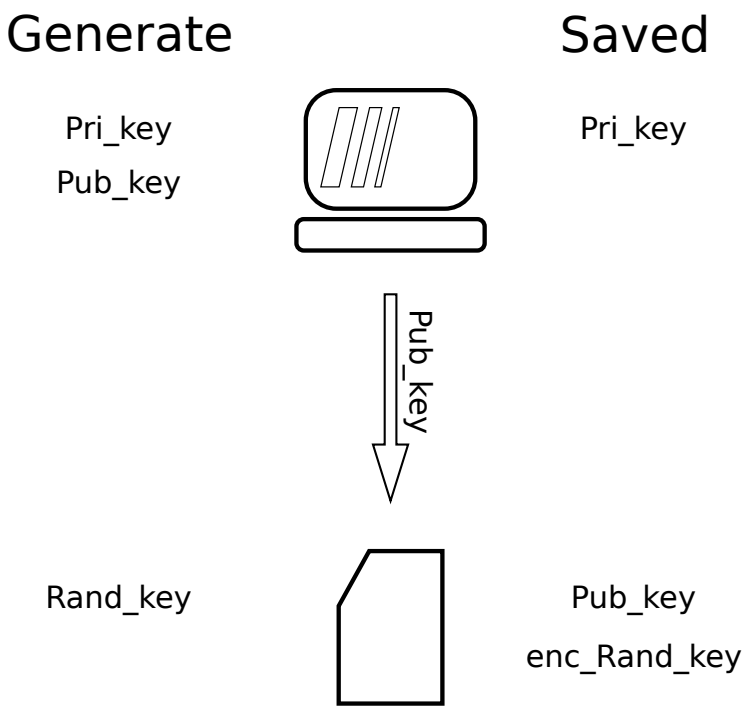

Figure 1: This illustration shows where the keys are generated and where they are stored

2.2 Encryption

The Rand_key[1] is then used as the key input in the chacha cipher to encrypt the data before it is saved to the memory see Figure 2. When the system is powered off the Rand_key that is only stored in the FPGA disappears due to the nature of an FPGA. The generation and encryption of the Rand_key is repeated for each session. So there will be multiple enc_Rand_key:s stored on the memory one for each session.

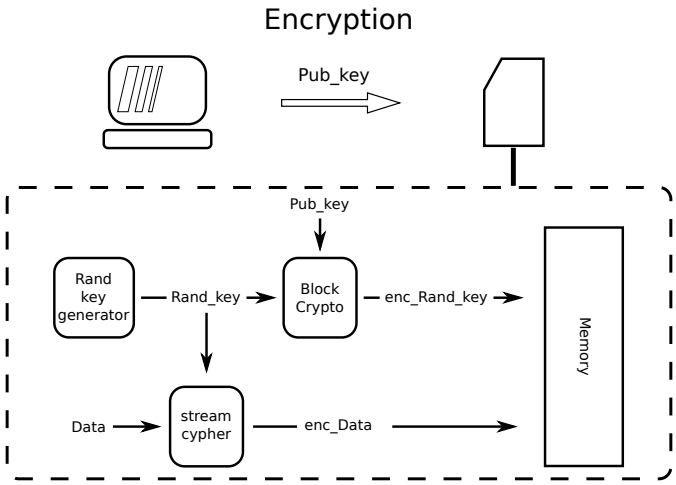

Figure 2: Basic data flow in encryption step

[1]This is the non encrypted random key.

2.3 Decryption

When decrypting the data one extracts the encrypted data and the Encrypted Random generated session Key (enc_Rand_key) on to the safe device were the Pri_key is saved. Then we use the Pri_key to decrypt the enc_Rand_key:s and then we can use the Rand_key:s to decrypt the data see Figure 3.

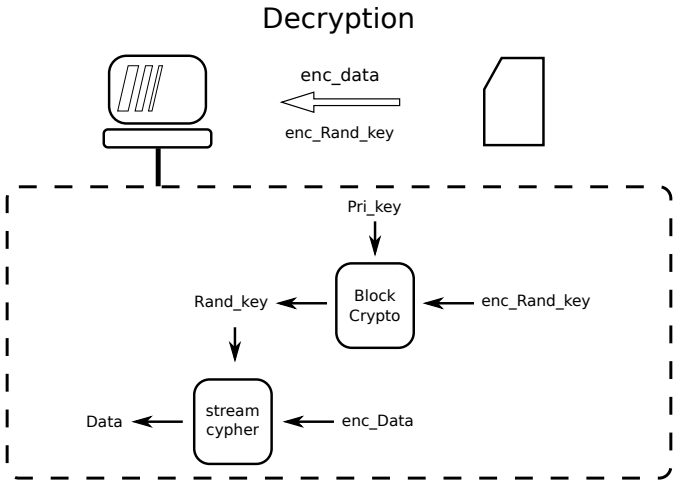

Figure 3: basic data flow in decryption step

3 Hardware

The hardware was intended to be an Printed Circuit Board (PCB) with an Field Programmable Gate Array (FPGA) with an memory and some support component. The FPGA that we use is the Artix7 t35.

4 sd slave progress

The SD slave comes form the Google vault project [2]. This part of the project does not work yet.

We have extracted the blocks that we think are needed for the project. There is a test bench to test and send commands to the design and see what responses the device sends back. So far we can read and write from the first sector in the memory but not to any other sector.

Function	Setting
Manufacturer	Spansion
Settings	Value
Density	256
Type	qspi
Width	x4-single

Table 1: This table includes settings for configuring the memory on the Arty z7 board.

4.1 Development environment

Here I will state the steps to get the debugging platform working. The system is designed for the Arty z7 board.

First start the Vivado project. Second we need to set the board to jtag boot mode this is done with an jumper on the board. Follow this with a power cycle. Open hardware manager and open target. Right click on the FPGA to add configuration memory device[2].

This produces two results we pick the *3.3V* version and press "ok". Vivado now asks if we want to program it we say "ok". Now we need to pick the *.elf* and *boot.bin* files. The boot.bin lives in the root of the project. And the *fsbl.elf* file resides in `*.sdk\fsbl\Debug`. Then program. Set boot jumper to spi mode and power cycle.

Then I used minicom[3] to connect to the system baud rate 115200. mmc info shuld give info about the device. Use `read mmc 0 0 1` to read first sector and `write mmc 0 0 1` to write first sector.

5 chacha

The stream cipher that used for the data encryption is called chacha [1] developed by Joachim Strömbergson [3]. We have made an axi ip block out of it, one stream port for the data and one address port for the setting example iv and key see Figure 4. There is an test bench that show how the axi block is intended to be used.

6 Helpful data

This section will list information that has been collected during the project that might help further development.

[2]If this does not work it might mean that you might need to remove an old one, right click on the memory and choose remove

[3]minicom is an text based terminal emulator

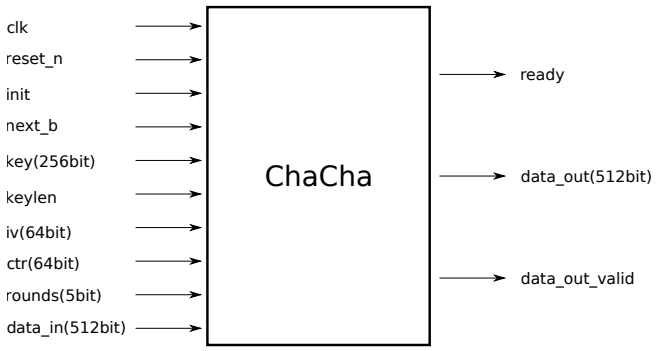

Figure 4: Port configuration chacha cipher

6.1 CSD list

During the debugging we listed the CSD values that was used in the gv_sd_slave we also made an list of the other CSD and CID values. We list the CSD list 1.0, 2.0 and the CID list also the original CSD settings from the Google Vault.

	This is the list from part1 Version (1.0)	
Name	Descripton	CSD-slice
CSD structure	CSD structure	[127:126]
-	reserved	[125:120]
TAAC	data read access-time-1	[119:112]
NSAC	data read access-time-2 in CLK cycles (NSAC * 100)	[111:104]
TRAN_SPEED	max. read data block length	[103:96]
CCC	card command classes	[95:84]
READ_BL_LEN	max. read data block length	[83:80]
READ_BL_PARTIAL	partial blocks for read allowed	[79:79]
WRITE_BLK_MISALIGN	write block misalignment	[78:78]
READ_BLK_MISALIGN	read block misalignment	[77:77]
DSR_IMP	DSR implemented	[76:76]
-	reserved	[75:74]
C_SIZE	device size	[73:62]
VDD_R_CURR_MIN	max.read current @ VDD min	[61:59]
VDD_R_CURR_MAX	max.read current @ VDD max	[58:56]
VDD_W_CURR_MIN	max.write current @ VDD min	[55:53]
VDD_W_CURR_MAX	max.write current @ VDD max	[52:50]
C_SIZE_MULT	device size multiplier	[49:47]
ERASE_BLK_EN	erase single block enable	[46:46]
SECTOR_SIZE	erase sector size	[45:39]
WP_GRP_SIZE	write protect group size	[38:32]
WP_GRP_ENABLE	write protect proup enable	[31:31]
-	reserved	[30:29]
R2W_FACTOR	write speed factor	[28:26]
WRITE_BL_LEN	max. write data block length	[25:22]
WRITE_BL_PARTIAL	partial blocks for write allowed	[21:21]
-	reserved	[20:16]
FILE_FORMAT_GRP	file format group	[15:15]
COPY	copy flag	[14:14]
PERM_WRITE_PROTECT	premanet write protection	[13:13]
TMP_WRITE_PROTECT	temporary write protection	[12:12]
FILE_FORMAT_GRP	file format	[11:10]
-	reserved	[9:8]
CRC	crc	[7:1]
-	not used always '1'	[0:0]

Table 2: Table shows CSD list Version 1.0

	This is the list from part1 Version (2.0)	
Name	Descripton	CSD-slice
CSD structure	CSD structure	[127:126]
-	reserved	[125:120]
TAAC	data read access-time-1	[119:112]
NSAC	data read access-time-2 in CLK cycles (NSAC * 100)	[111:104]
TRAN_SPEED	max. read data block length	[103:96]
CCC	card command classes	[95:84]
READ_BL_LEN	max. read data block length	[83:80]
READ_BL_PARTIAL	partial blocks for read allowed	[79:79]
WRITE_BLK_MISALIGN	write block misalignment	[78:78]
READ_BLKe_MISALIGN	read block misalignment	[77:77]
DSR_IMP	DSR implemented	[76:76]
-	reserved	[75:70]
C_SIZE	device size	[69:48]
	reserved	[47:47]
ERASE_BLK_EN	erase single block enable	[46:46]
SECTOR_SIZE	erase sector size	[45:39]
WP_GRP_SIZE	write protect group size	[38:32]
WP_GRP_ENABLE	write protect proup enable	[31:31]
-	reserved	[30:29]
R2W_FACTOR	write speed factor	[28:26]
WRITE_BL_LEN	max. write data block length	[25:22]
WRITE_BL_PARTIAL	partial blocks for write allowed	[21:21]
-	reserved	[20:16]
FILE_FORMAT_GRP	file format group	[15:15]
COPY	copy flag	[14:14]
PERM_WRITE_PROTECT	premanet write protection	[13:13]
TMP_WRITE_PROTECT	temporary write protection	[12:12]
FILE_FORMAT_GRP	file format	[11:10]
-	reserved	[9:8]
CRC	crc	[7:1]
-	not used always '1'	[0:0]

Table 3: Table shows CSD list Version 2.0

	Settings from GV sd_parms.vh		
Name	Value		Comment
CSD_CSD_STRUCTURE	b01		
CSD_TAAC	h0E		
CSD_NSAC	h00		
CSD_TRAN_SPEED_25	h32		
CSD_TRAN_SPEED_50	h5A		
CSD_CCC	b01_1_110110101		
CSD_READ_BL_LEN	9		512 byte Blocks
CSD_READ_BL_PARTIAL	h0		Partial block reads not implemented
CSD_WRITE_BLK_MISALIGN	h0		Write misalingned block not implemented
CSD_READ_BLK_MISALIGN	h0		Read misaligned block not implemented
CSD_DSR_IMPL	h0		DSR not implemented
CSD_C_SIZE	8		4.5 MBYTE
SD_TOTAL_BLOCKS	(CSD_C_SIZE+1) * 1024	9216	
CSD_ERASE_BLK_EN	h1		
CSD_SECTOR_SIZE	h7F		
CSD_WP_GRP_SIZE	h0		
CSD_WP_GRP_ENABLE	h0		
CSD_R2W_FACTOR	b010		
CSD_WRITE_BL_LEN	h9		
CSD_WRITE_BL_PARTIAL	h0		
CSD_FILE_FORMAT_GRP	h0		
CSD_COPY	h0		
CSD_PERM_WRITE_PROTECT	h0		
CSD_TMP_WRITE_PROTECT	h0		
CSD_FILE_FORMAT	b00		
	Settings from GV sd_const.vh		
Name	Value		
CMD9_SEND_CSD	d9		
CMD27_PROGRAM_CSD	d27		
STAT_CSD_OVERWRITE	d16		

Table 4: This is the CSD settings that was originally in the Google Vault project.

Name	Field	Width	CID-slice
Manufactire ID	MID	8	[127:120]
OEM/Application ID	OID	16	[119:104]
Product name	PNM	40	[103:64]
Prodict revision	PRV	8	[63:56]
Product serial number	PSN	32	[55:24]
reserved	–	4	[23:20]
Manufacturing date	MDT	12	[19:8]
CRC7 checksum	CRC	7	[7:1]
"not used, always 1"	-	1	[0:0]

Table 5: This table shows the CID list

7 Whishlist

Here we will list some features that we would like to have in the future. Theese are extras but good to have.

7.1 Hidden files

We would like to have the possibility to hide the files so that every time you start up the memory card it looks empty. This would be done by making an temporary FAT table that is used during the session then hidden between the MBS and the real FAT table.

7.2 See pictures when camera is still on

We would like to be able to look at the pictures during the session. This would mean that the cipher needs to work in both directions during the session.

7.3 Fake pictures

We would also like to be able to prep the memory card with fake data in the camera case that would be preloaded with pictures. These would show up if someone would inspect the card. This would also be helped by the see pictures while session is active if the system is tested.

Glossary

cipher Is an mathematical algorithm that takes two inputs data and an key and outputs an new data string that can not be read whit out an corresponding key. 3

gv_sd_slave This is the sd_slave from the google vault project. 7

key(s) This refers to an sires of bits used as an input to the cipher. 3

MBS Master boot record is an boot sector in the beginning of an memory that tells the system where to start. 12

Acronyms

enc_Rand_key Encrypted Random generated session Key. 6

FPGA Field Programmable Gate Array. 6

PCB Printed Circuit Board. 6

Pri_key private key. 3

Pub_key Public key. 3

Rand_key Random generated session Key. 3

References

[1] ChaCha. `https://git.cryptech.is/core/cipher/chacha.git`. Accessed: 2018-05-22.

[2] GoogleVault. `https://github.com/ProjectVault/orp`. Accessed: 2018-05-22.

[3] joachim strömbergson. Chacha (cypher).