





## **Dynamic Memory in Today's Systems**



Dynamic memory, such as SDRAM (Synchronous Dynamic Random Access Memory), DDR ,DDR2, DDR3 (Double Data Rate SDRAM Type 1,2,3), and various types of GDDR (Graphics DDR) are ubiquitous in current systems. Mostly used as main memory for the CPU, it is also used as cache in hard disk drive, for GPUs and storage controllers. Even in small embedded systems, smart phones, or TV receivers, it is used as temporal storage.

Unique identifiers and true random numbers are crucial in many cryptographic applications. One way to generate such numbers are so called Physically Unclonable Functions (PUF). These functions exploit variations in the fabrication of semiconductor devices to generate such patterns. Last year, we proposed a method of using Dynamic Memory for PUF. If standard DRAM, which is present in such a variety of devices, can be used as PUF, each device can generate its own, unique identifier without requiring any additional hardware.

The goal was to find out, whether commercial off-the-shelf Dynamic Memory can be used to generate the required signatures to calculate the identifiers. Further, we wanted to investigate the stability of the data patterns in terms of temperature and life time.

# **DRAM PUF**

### PUFs proposed so far:

- Race conditions: delay differences in "equal" pairs of signal paths
- Ring oscillators: frequency variations
- Static RAM (SRAM): power-up pattern
- **Optical:** light propagation in passivation layer to on-chip photo diodes

### Our proposal:

Dynamic RAM PUF (DRAM PUF)  $\rightarrow$ 

	unclonable	inputs	outputs	random
Race conditions	yes	some	few	(yes)
Ring oscillators	yes	some	few	(yes)
Optical	ves	some	few	(ves)

# **DRAM PUF Operation**

The PUF operation conducts the following tasks:

- write pattern = PUF input (raw)
- state is stored on capacitors
- refresh is disabled
- leakage (de)charges capacitors physical variations
- read word(s) = PUF output (raw)
- sense amplifiers discriminate 0|1 physical variations

When retrieving the node charge, timing is an important factor. The reliability of the output pattern of the PUF directly depends on that

## DRAM torage array BLm WLn ---RAM cell conductio interconne WLn+1 ••

# **Investigating DDR3 Memory**



To test the functionality and behavior of commercial DRAM, a test system has been developed and set up.

The test system hosts the following components:

- Xilinx ML605 evaluation board with
  - Xilinx Virtex 6 FPGA
  - DDR3 SO-DIMM slot
  - 1 Gbit/s Ethernet interface

SRAM	yes	none	many	no
DRAM	yes	many	many	yes

Using DRAM as a PUF circuit has some advantages over the other implementations. The most significant one is the large input space. An arbitrary input pattern can be written to the memory array and a corresponding output pattern can be gathered which is, ideally, statistically independent of the input pattern.

#### For DRAM PUFs, a variety of DRAMs can be applicable:

- Standard DRAM
- **Embedded DRAM**
- **Dynamic Flip Flops**
- Gain Cells

timing. If the storage nodes are read out too early, almost no changes have occurred. If we wait too long, the charges have vanished and no information can be extracted. Therefore, the optimal time window has to be found after every startup and even during normal operation.



### Initialization: find wait time write pattern

- wait t(k++)
- read pattern

repeat

until 25% cells toggled

#### PUF readout: evaluate function

- (input pre-processing)
- write pattern
- wait t<sub>25%</sub>
  - read word(s)
  - (output post-processing)

Even though Xilinx provides a DDR3 memory controller, we had to build our own version of a DDR3 controller. The Xilinx controller was developed for high throughput and performance. However, there was no possibility to disable the refresh of the RAM. Therefore, we implemented an own version of a DDR3 controller which allows us to directly manipulate all the commands and RAM data which are sent to the memory module.

The test program works as follows:

- Write data to RAM
- Wait
- Read data from RAM
- Compare with written data

There are two different modes implemented for the waiting. In one mode, the memory is idle during waiting. In the other, the memory is constantly being written to, except to the locations, where the test data is stored. Furthermore, we can choose between random data, all zeros and all ones to be written to the memory. For all our tests, 64 Kbytes (512 Kbits) of data have been written.



# Conclusions

DDR memory shows all the properties which are required for a PUF functionality. However, in terms of speed, they are not very suitable. The decay rate is astonishingly low compared to the minimum refresh cycle time of 64ms. Since DDR Memories are designed to be fully reliable in environments where temperatures of above 60°C can be reached, a large enough margin had to be implemented. Our working conditions were far beneath the worst case conditions. The second reason for the obtained results can be the restrictions of our design. To reduce the designing effort, the whole test system runs at the lowest frequency which is defined in the DDR specifications (250 MHz). Increasing the frequency would lead to a higher power dissipation and higher temperature of the module and increase the decay.



As can be seen in the left graph, the loss of data is very slow. After 64s, only 20 bits have been flipped. After 4096s, more than one hour, only 6% (31874) of the bits have flipped. Writing data while waiting increased the number of bits which have toggled by a factor of 1.5. These tests have been conducted at room temperature. The temperature of the RAM modules have been measured to be 32°C.

If the temperature of the memory is increased by only 6°C to 38°C, the decay rate could be increased by a factor of 1.6. After 4096s, 51184 bits (9.7%) have flipped.

The retrieved data patterns proved to be very stable. After several runs with the same settings (waiting time, random data, write enabled while waiting), we always extracted the same data patterns.

When comparing various modules, we could see, that their responses have only a few bits in common that behave the same. E.g. after 256s, only 8 bit locatoins showed the same behavior on both modules, while random 630 bits respectively 629 bits toggled in the two modules.

Principal Investigator: Nicolas Gisin (Nicolas.Gisin@unige.ch)

Contact: Norbert Felber (felber@iis.ee.ethz.ch)