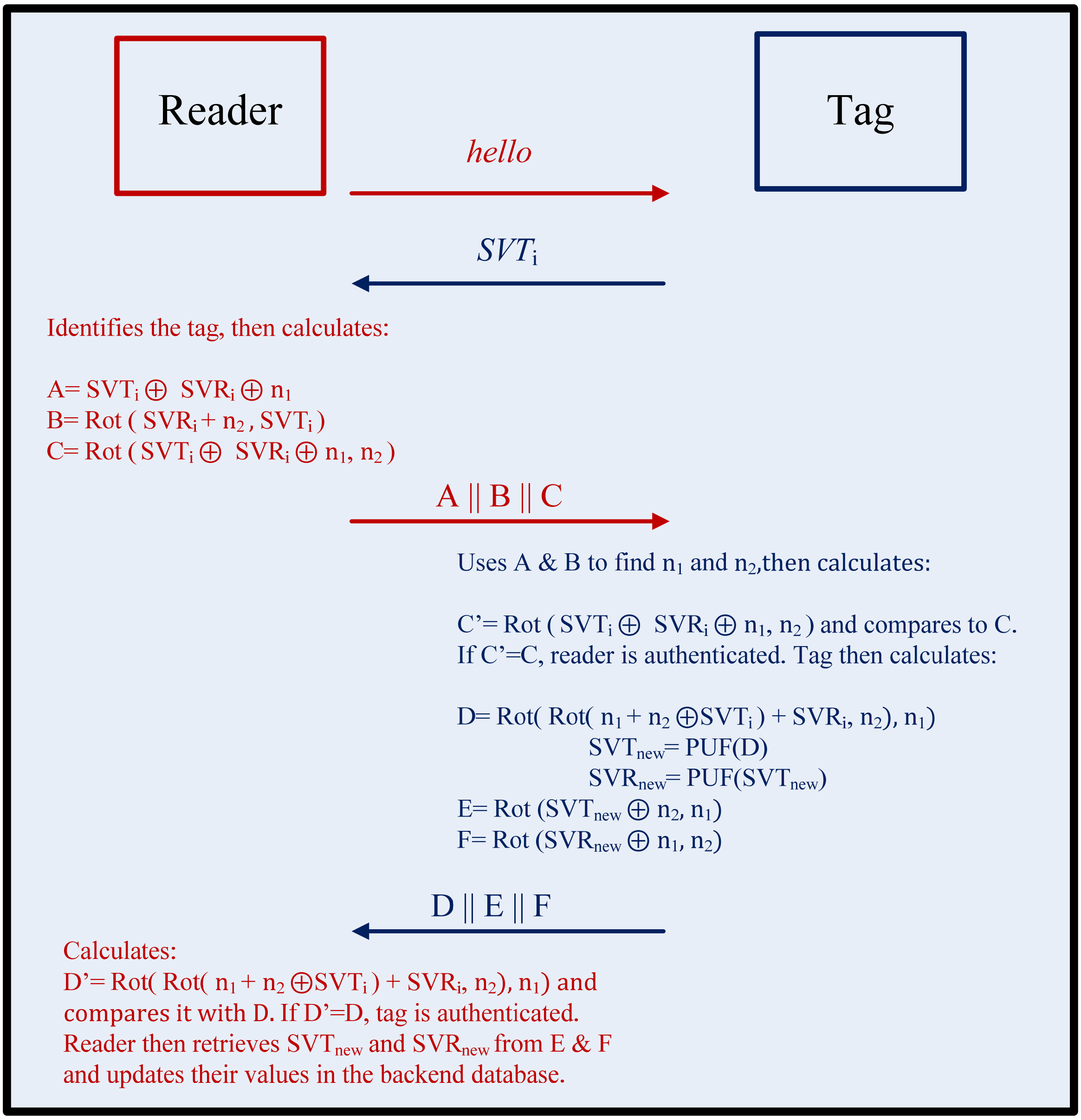

Reader
Tag
hello
SVT_i
Identifies the tag, then calculates:
$A = SVT_i \oplus SVR_i \oplus n_1$
$B = Rot(SVR_i + n_2, SVT_i)$
$C = Rot(SVT_i \oplus SVR_i \oplus n_1, n_2)$
A || B || C
Uses A & B to find n_1 and n_2,then calculates:
$C' = Rot(SVT_i \oplus SVR_i \oplus n_1, n_2)$ and compares to C.
If C'=C, reader is authenticated. Tag then calculates:
$D = Rot(Rot(n_1 + n_2 \oplus SVT_i) + SVR_i, n_2), n_1)$
$SVT_{new} = PUF(D)$
$SVR_{new} = PUF(SVT_{new})$
$E = Rot(SVT_{new} \oplus n_2, n_1)$
$F = Rot(SVR_{new} \oplus n_1, n_2)$
D || E || F
Calculates:
$D' = Rot(Rot(n_1 + n_2 \oplus SVT_i) + SVR_i, n_2), n_1)$ and compares it with D. If D'=D, tag is authenticated.
Reader then retrieves SVT_{new} and SVR_{new} from E & F and updates their values in the backend database.

Figure 1. Proposed Protocol

6. Performance Analysis

The performance and storage requirements of the protocol are studied in this section.

- Computational Cost: All the operations used in this protocol are compliant with ultra-lightweight tags and can be very efficiently implemented in hardware. According to [13], the implementation of the PUF function requires six to eight gates for each input bit; thus a 96-bit PUF function will require at most 768 gates. This means that PUFs are a much better solution than using hash functions that require at least 3500 gates to be implemented. Furthermore, the rotation operation is also lightweight and can be implemented easily on ultra-lightweight tags.

- Storage Requirements: Each tag needs to store two couples of SVT_i and SVR_i , old and updated values, in addition to n_1. Each of these identifiers has a length of 96 bits in compliance with EPCGlobal. Additionally, the tag needs to store a maximum of five intermediate 96-bit values during the authentication phase. All of these values are stored in a rewritable memory because they change during different authentication sessions. So the total storage requirement on the tag is:

$$10*96 = 960 \text{ bits}$$

- Communication Cost: Assuming that the “hello” message is 5 bytes and knowing that messages A through F and SVT_i all have a length of 96 bits, the total communication cost of this protocol is:

  7*96 +5*8= 712 bits

Table 1 compares the performance of the different ultra-lightweight protocols surveyed in this work.

Table 1. Performance Comparison

Protocol	Storage Req.(bits)	Communication Cost (bits)
Gossamer	960	520
LMAP	1056	520
SASI	864	520
Work in [12]	864	424
Proposed Work	960	712

The communication cost in this work could be significantly decreased if the update phase is performed separately in the tag and in the reader, but then the updated values would not involve the use of PUFs and hence there is a tradeoff between uniqueness of updated values and communication cost.

7. Implementation

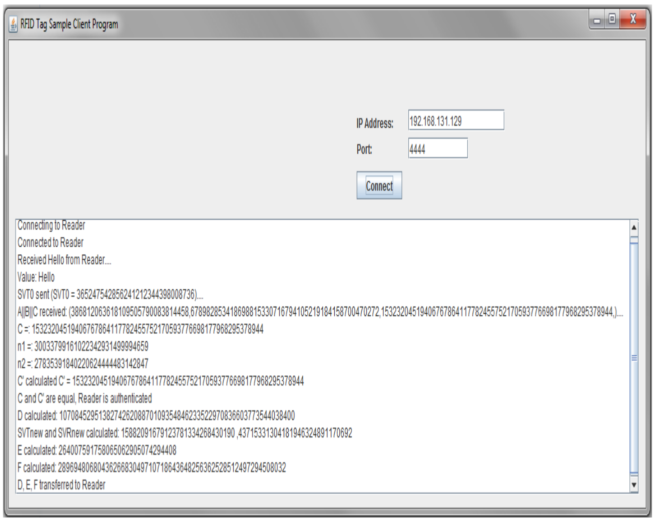
In this section we describe a sample implementation of the security protocol described in Figure 1. We simulated the operation the RFID tag and reader using a client/server program developed in Java. In this simulation model the RFID tag plays the role of the client while the reader assumes the role of the server. The network interaction is realized using Java sockets that abstract a TCP client/server connection. The reader waits for tag connections on a specified IP address and port. Once an RFID tag successfully establishes a connection with the reader, the protocol steps, as specified in Figure 1, are executed resulting in a mutual authentication between the RFID tag and reader. We believe that this simulation model presents a viable proof of concept that demonstrates the correctness of the protocol before future deployment on real RFID tags and readers. The NetBeans v7.0.1 Integrated Development Environment (IDE) [14] is used for developing the client/server program using the Java 7 platform [15]. The server machine runs Ubuntu Linux v 9.04 and having the following hardware specifications:

- Processors: Intel(R) Core(TM) i7 CPU Q 720 running at 1.6 GHz
- Memory: 4GB RAM

The client machine runs Windows 7 with the following hardware specifications:

- Processors: Two Intel(R) Xeon CPUs running at 3.8 GHz
- Memory: 2 GB RAM

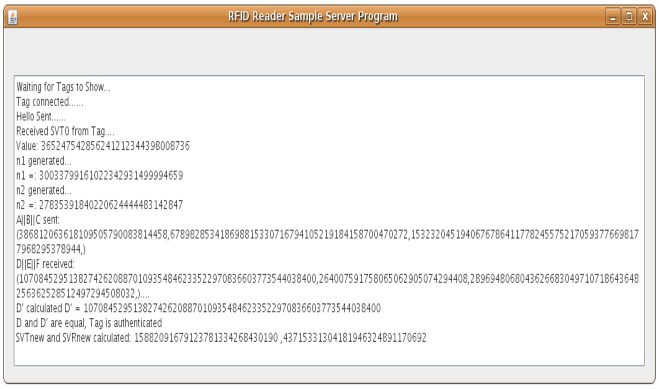
The snapshots presented in Figures 2 and 3 respectively demonstrate a sample execution of the RFID tag and reader protocol steps. The diagnostic messages provided by the client and server programs indicate the successful execution of the different protocol steps and the values of the parameters involved in the mutual authentication procedure. To test the validity of the protocol exchanged messages and their full compliance with the protocol specifications; we deployed the Wireshark network protocol analyzer [16] on the server machine to capture the inbound and outbound protocol packets exchanged by the server network interface. Figure 4 shows the details of the “*hello*” packet sent from the reader to the tag as captured by Wireshark. Figure 5 represents the tag to reader message containing the value of “*SVT0*”. The Wireshark packet representation of Figures 6 demonstrates the message containing the values of the *A*, *B*, and *C* parameters sent from the reader to the tag. Similarly, the packet representation of Figure 7 indicates the message containing the values of the *D*, *E*, and *F* parameters sent from the tag to the reader.



IP Address: 192.168.131.129
Port: 4444

```
Connecting to Reader
Connected to Reader
Received Hello from Reader....
Value: Hello
SVT0 sent (SVT0 = 365247542856241212344398008736)....
A||B||C received: (386812063618109505790083814458,678982853418698815330716794105219184158700470272,15323204519406767864117782455752170593776698177968295378944,)....
C =: 15323204519406767864117782455752170593776698177968295378944
n1 =: 30033799161022342931499994659
n2 =: 27835391840220624444483142847
C' calculated C' = 15323204519406767864117782455752170593776698177968295378944
C and C' are equal, Reader is authenticated
D calculated: 107084529513827426208870109354846233522970836603773544038400
SVTnew and SVRnew calculated: 15882091679123781334268430190 ,43715331304181946324891170692
E calculated: 264007591758065062905074294408
F calculated: 2896948068043626683049710718643648256362528512497294508032
D, E, F transferred to Reader
```

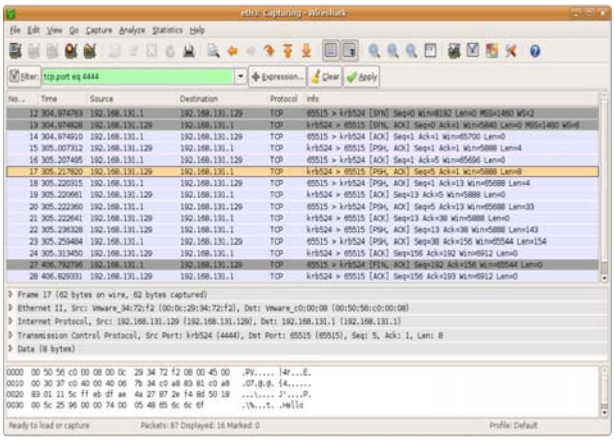
Figure 2. Client-side program executing the tag protocol.



```
Waiting for Tags to Show...
Tag connected......
Hello Sent......
Received SVT0 from Tag....
Value: 3652475428562412123443980087 36
n1 generated...
n1 =: 3003379916102234293149999 4659
n2 generated...
n2 =: 2783539184022062444448314 2847
A||B||C sent:
(386812063618109505790083814458,678982853418698815330716794105219184158700470272,1532320451940676786411778245575217059377669817
7968295378944,)
D||E||F received:
(107084529513827426208870109354846233522970836603773544038400,264007591758065062905074294408,28969480680436266830497107186436 48
2563625285124972945 08032,)....
D' calculated D' = 107084529513827426208870109354846233522970836603773544038400
D and D' are equal, Tag is authenticated
SVTnew and SVRnew calculated: 158820916791237813342684301 90 ,4371533130418194632489117 0692
```

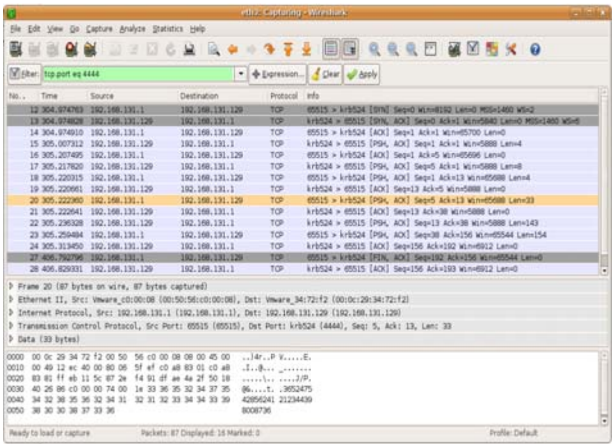
Figure 3. Snapshot of the server-side program executing the RFID reader protocol steps

By comparing the parameter values displayed by the tag and reader diagnostic messages (see Figures 2 and 3) and the packet contents captured by Wireshark (see Figures 4, 5, 6, and 7) we realize that the protocol sample implementation demonstrates a 100% compatibility with the protocol specification.



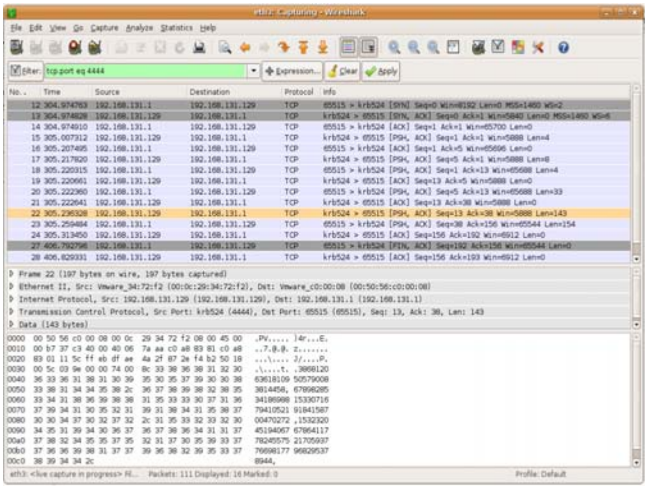
```
Filter: tcp.port eq 4444

No. .  Time        Source           Destination      Protocol  Info
  12 304.974763  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
  13 304.974828  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 WS=6
  14 304.974910  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [ACK] Seq=1 Ack=1 Win=65700 Len=0
  15 305.007312  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [PSH, ACK] Seq=1 Ack=1 Win=5888 Len=4
  16 305.207495  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [ACK] Seq=1 Ack=5 Win=65696 Len=0
  17 305.217820  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [PSH, ACK] Seq=5 Ack=1 Win=5888 Len=8
  18 305.220315  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [PSH, ACK] Seq=1 Ack=13 Win=65688 Len=4
  19 305.220661  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=13 Ack=5 Win=5888 Len=0
  20 305.222360  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [PSH, ACK] Seq=5 Ack=13 Win=65688 Len=33
  21 305.222641  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=13 Ack=38 Win=5888 Len=0
  22 305.236328  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [PSH, ACK] Seq=13 Ack=38 Win=5888 Len=143
  23 305.259484  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [PSH, ACK] Seq=38 Ack=156 Win=65544 Len=154
  24 305.313450  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=156 Ack=192 Win=6912 Len=0
  27 406.792796  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [FIN, ACK] Seq=192 Ack=156 Win=65544 Len=0
  28 406.829331  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=156 Ack=193 Win=6912 Len=0

▷ Frame 17 (62 bytes on wire, 62 bytes captured)
▷ Ethernet II, Src: Vmware_34:72:f2 (00:0c:29:34:72:f2), Dst: Vmware_c0:00:08 (00:50:56:c0:00:08)
▷ Internet Protocol, Src: 192.168.131.129 (192.168.131.129), Dst: 192.168.131.1 (192.168.131.1)
▷ Transmission Control Protocol, Src Port: krb524 (4444), Dst Port: 65515 (65515), Seq: 5, Ack: 1, Len: 8
▷ Data (8 bytes)

0000  00 50 56 c0 00 08 00 0c  29 34 72 f2 08 00 45 00   .PV..... )4r...E.
0010  00 30 37 c0 40 00 40 06  7b 34 c0 a8 83 81 c0 a8   .07.@.@. {4......
0020  83 01 11 5c ff eb df ae  4a 27 87 2e f4 8d 50 18   ...\.... J'....P.
0030  00 5c 25 96 00 00 74 00  05 48 65 6c 6c 6f         .\%...t. .Hello

Ready to load or capture    Packets: 87 Displayed: 16 Marked: 0    Profile: Default
```

Figure 4. Wireshark capture showing the "hello" message



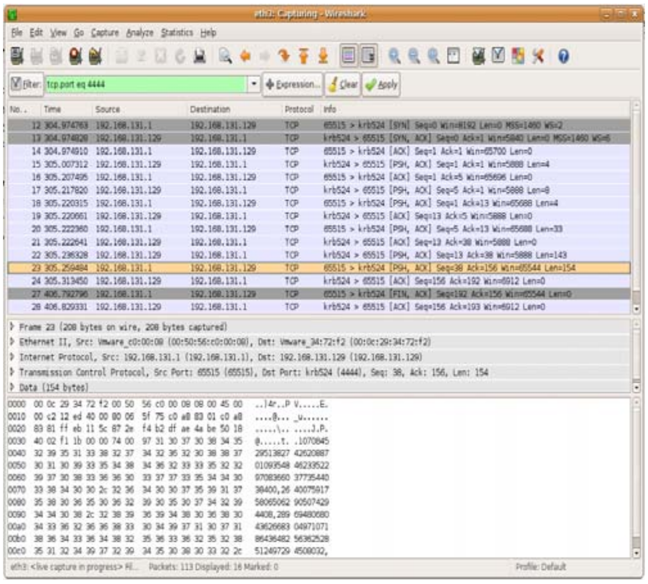
```
Filter: tcp.port eq 4444

No. .  Time        Source           Destination      Protocol  Info
  12 304.974763  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
  13 304.974828  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 WS=6
  14 304.974910  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [ACK] Seq=1 Ack=1 Win=65700 Len=0
  15 305.007312  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [PSH, ACK] Seq=1 Ack=1 Win=5888 Len=4
  16 305.207495  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [ACK] Seq=1 Ack=5 Win=65696 Len=0
  17 305.217820  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [PSH, ACK] Seq=5 Ack=1 Win=5888 Len=8
  18 305.220315  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [PSH, ACK] Seq=1 Ack=13 Win=65688 Len=4
  19 305.220661  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=13 Ack=5 Win=5888 Len=0
  20 305.222360  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [PSH, ACK] Seq=5 Ack=13 Win=65688 Len=33
  21 305.222641  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=13 Ack=38 Win=5888 Len=0
  22 305.236328  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [PSH, ACK] Seq=13 Ack=38 Win=5888 Len=143
  23 305.259484  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [PSH, ACK] Seq=38 Ack=156 Win=65544 Len=154
  24 305.313450  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=156 Ack=192 Win=6912 Len=0
  27 406.792796  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [FIN, ACK] Seq=192 Ack=156 Win=65544 Len=0
  28 406.829331  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=156 Ack=193 Win=6912 Len=0

▷ Frame 20 (87 bytes on wire, 87 bytes captured)
▷ Ethernet II, Src: Vmware_c0:00:08 (00:50:56:c0:00:08), Dst: Vmware_34:72:f2 (00:0c:29:34:72:f2)
▷ Internet Protocol, Src: 192.168.131.1 (192.168.131.1), Dst: 192.168.131.129 (192.168.131.129)
▷ Transmission Control Protocol, Src Port: 65515 (65515), Dst Port: krb524 (4444), Seq: 5, Ack: 13, Len: 33
▷ Data (33 bytes)

0000  00 0c 29 34 72 f2 00 50  56 c0 00 08 08 00 45 00   ..)4r..P V.....E.
0010  00 49 12 ec 40 00 80 06  5f ef c0 a8 83 01 c0 a8   .I..@... _.......
0020  83 81 ff eb 11 5c 87 2e  f4 91 df ae 4a 2f 50 18   .....\.. ....J/P.
0030  40 26 86 c0 00 00 74 00  1e 33 36 35 32 34 37 35   @&....t. .3652475
0040  34 32 38 35 36 32 34 31  32 31 32 33 34 34 33 39   42856241 21234439
0050  38 30 30 38 37 33 36                               8008736

Ready to load or capture    Packets: 87 Displayed: 16 Marked: 0    Profile: Default
```

Figure 5. Wireshark capture of the packet representing the "SVT0" message from the tag to the reader



```
Filter: tcp.port eq 4444

No. .  Time        Source           Destination      Protocol  Info
  12 304.974763  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
  13 304.974828  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 WS=6
  14 304.974910  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [ACK] Seq=1 Ack=1 Win=65700 Len=0
  15 305.007312  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [PSH, ACK] Seq=1 Ack=1 Win=5888 Len=4
  16 305.207495  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [ACK] Seq=1 Ack=5 Win=65696 Len=0
  17 305.217820  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [PSH, ACK] Seq=5 Ack=1 Win=5888 Len=8
  18 305.220315  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [PSH, ACK] Seq=1 Ack=13 Win=65688 Len=4
  19 305.220661  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=13 Ack=5 Win=5888 Len=0
  20 305.222360  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [PSH, ACK] Seq=5 Ack=13 Win=65688 Len=33
  21 305.222641  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=13 Ack=38 Win=5888 Len=0
  22 305.236328  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [PSH, ACK] Seq=13 Ack=38 Win=5888 Len=143
  23 305.259484  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [PSH, ACK] Seq=38 Ack=156 Win=65544 Len=154
  24 305.313450  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=156 Ack=192 Win=6912 Len=0
  27 406.792796  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [FIN, ACK] Seq=192 Ack=156 Win=65544 Len=0
  28 406.829331  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=156 Ack=193 Win=6912 Len=0

▷ Frame 22 (197 bytes on wire, 197 bytes captured)
▷ Ethernet II, Src: Vmware_34:72:f2 (00:0c:29:34:72:f2), Dst: Vmware_c0:00:08 (00:50:56:c0:00:08)
▷ Internet Protocol, Src: 192.168.131.129 (192.168.131.129), Dst: 192.168.131.1 (192.168.131.1)
▷ Transmission Control Protocol, Src Port: krb524 (4444), Dst Port: 65515 (65515), Seq: 13, Ack: 38, Len: 143
▷ Data (143 bytes)

0000  00 50 56 c0 00 08 00 0c  29 34 72 f2 08 00 45 00   .PV..... )4r...E.
0010  00 b7 37 c3 40 00 40 06  7a aa c0 a8 83 81 c0 a8   ..7.@.@. z.......
0020  83 01 11 5c ff eb df ae  4a 2f 87 2e f4 b2 50 18   ...\.... J/....P.
0030  00 5c 03 9e 00 00 74 00  8c 33 38 36 38 31 32 30   .\....t. .3868120
0040  36 33 36 31 38 31 30 39  35 30 35 37 39 30 30 38   63618109 50579008
0050  33 38 31 34 34 35 38 2c  36 37 38 39 38 32 38 35   3814458, 67898285
0060  33 34 31 38 36 39 38 38  31 35 33 33 30 37 31 36   34186988 15330716
0070  37 39 34 31 30 35 32 31  39 31 38 34 31 35 38 37   79410521 91841587
0080  30 30 34 37 30 32 37 32  2c 31 35 33 32 33 32 30   00470272 ,1532320
0090  34 35 31 39 34 30 36 37  36 37 38 36 34 31 31 37   45194067 67864117
00a0  37 38 32 34 35 35 37 35  32 31 37 30 35 39 33 37   78245575 21705937
00b0  37 36 36 39 38 31 37 37  39 36 38 32 39 35 33 37   76698177 96829537
00c0  38 39 34 34 2c                                     8944,

eth3: <live capture in progress> Fil...    Packets: 111 Displayed: 16 Marked: 0    Profile: Default
```

Figure 6. Wireshark capture of the packet representing the "A||B||C" message from the reader to the tag.



```
Filter: tcp.port eq 4444

No. .  Time        Source           Destination      Protocol  Info
  12 304.974763  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
  13 304.974828  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 WS=6
  14 304.974910  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [ACK] Seq=1 Ack=1 Win=65700 Len=0
  15 305.007312  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [PSH, ACK] Seq=1 Ack=1 Win=5888 Len=4
  16 305.207495  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [ACK] Seq=1 Ack=5 Win=65696 Len=0
  17 305.217820  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [PSH, ACK] Seq=5 Ack=1 Win=5888 Len=8
  18 305.220315  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [PSH, ACK] Seq=1 Ack=13 Win=65688 Len=4
  19 305.220661  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=13 Ack=5 Win=5888 Len=0
  20 305.222360  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [PSH, ACK] Seq=5 Ack=13 Win=65688 Len=33
  21 305.222641  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=13 Ack=38 Win=5888 Len=0
  22 305.236328  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [PSH, ACK] Seq=13 Ack=38 Win=5888 Len=143
  23 305.259484  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [PSH, ACK] Seq=38 Ack=156 Win=65544 Len=154
  24 305.313450  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=156 Ack=192 Win=6912 Len=0
  27 406.792796  192.168.131.1    192.168.131.129  TCP  65515 > krb524 [FIN, ACK] Seq=192 Ack=156 Win=65544 Len=0
  28 406.829331  192.168.131.129  192.168.131.1    TCP  krb524 > 65515 [ACK] Seq=156 Ack=193 Win=6912 Len=0

▷ Frame 23 (208 bytes on wire, 208 bytes captured)
▷ Ethernet II, Src: Vmware_c0:00:08 (00:50:56:c0:00:08), Dst: Vmware_34:72:f2 (00:0c:29:34:72:f2)
▷ Internet Protocol, Src: 192.168.131.1 (192.168.131.1), Dst: 192.168.131.129 (192.168.131.129)
▷ Transmission Control Protocol, Src Port: 65515 (65515), Dst Port: krb524 (4444), Seq: 38, Ack: 156, Len: 154
▷ Data (154 bytes)

0000  00 0c 29 34 72 f2 00 50  56 c0 00 08 08 00 45 00   ..)4r..P V.....E.
0010  00 c2 12 ed 40 00 80 06  5f 75 c0 a8 83 01 c0 a8   ....@... _u......
0020  83 81 ff eb 11 5c 87 2e  f4 b2 df ae 4a be 50 18   .....\.. ....J.P.
0030  40 02 f1 1b 00 00 74 00  97 31 30 37 30 38 34 35   @.....t. .1070845
0040  32 39 35 31 33 38 32 37  34 32 36 32 30 38 38 37   29513827 42620887
0050  30 31 30 39 33 35 34 38  34 36 32 33 33 35 32 32   01093548 46233522
0060  39 37 30 38 33 36 36 30  33 37 37 33 35 34 34 30   97083660 37735440
0070  33 38 34 30 30 2c 32 36  34 30 30 37 35 39 31 37   38400,26 40075917
0080  35 38 30 36 35 30 36 32  39 30 35 30 37 34 32 39   58065062 90507429
0090  34 34 30 38 2c 32 38 39  36 39 34 38 30 36 38 30   4408,289 69480680
00a0  34 33 36 32 36 36 38 33  30 34 39 37 31 30 37 31   43626683 04971071
00b0  38 36 34 33 36 34 38 32  35 36 33 36 32 35 32 38   86436482 56362528
00c0  35 31 32 34 39 37 32 39  34 35 30 38 30 33 32 2c   51249729 4508032,

eth3: <live capture in progress> Fil...    Packets: 113 Displayed: 16 Marked: 0    Profile: Default
```

Figure 7. Wireshark capture of the packet representing the "D||E||F" message from the tag to the reader

8. Conclusion

This paper surveyed RFID systems in general stating their architecture and applications. Then, some recently proposed RFID security protocols were presented while focusing on the ultra-lightweight family as those comply with the limited processing power of the most commonly used passive tags. Building on that, a protocol that achieves mutual authentication for ultra-lightweight tags was proposed. The protocol comprises three main stages: tag identification, mutual authentication, and an update phase. It uses light operations and a PUF circuit that only requires about 3000 gated to be implemented and dedicated for security on passive tags. Finally, the security and performance of the proposed protocol were analyzed leading to the conclusion that the protocol offers immunity against a broad range of attacks while having an excellent performance.

9. References

[1] Garfinkel, S.L.; Juels, A.; Pappu, R.; , "RFID privacy: an overview of problems and proposed solutions," *Security & Privacy, IEEE* , vol.3, no.3, pp. 34- 43, May-June 2005.

[2] Boyeon Song and Chris J Mitchell, “RFID Authentication Protocol for Low-cost Tags,” *WiSec*, Alexandria, Virginia, March 31–April 2, 2008.

[3] Juels, A.; , "RFID security and privacy: a research survey," *IEEE Journal on Selected Areas in Communications,* vol.24, no.2, pp. 381-394, Feb. 2006.

[4] Wehbe, S.; Kayssi, A.; Chehab, A.; Elhajj, I.; , "Mutual Authentication Scheme for EPC Tags-Readers in the Supply Chain," *3rd International Conference on New Technologies, Mobility and Security (NTMS), 2009*, pp.1-5, 20-23 Dec. 2009.

[5] Shaoying Cai, Yingjiu Li, Tieyan Li, Robert H. Deng, “Attacks and Improvements to an RFID Mutual Authentication Protocol and its Extensions”, *WiSec*, Zurich, 2009.

[6] Yosef Oren, Martin Feldhofer, “A Low-Resource Public-Key Identification Scheme for RFID Tags and Sensor Nodes”, *WiSec,* Zurich, 2009.

[7] Y. K. Lee, Lejla Batina, Dave Singelée, Ingrid Verbauwhede, “Low-Cost Untraceable Authentication Protocols for RFID”, *WiSec,* New Jersey, 2010.

[8] Pedro Peris-Lopez, Julio Cesar Hernandez-Castro, Juan M.E. Tapiador, and Arturo Ribagorda, “LMAP: A Real Lightweight Mutual Authentication Protocol for Low-cost RFID tags”.

[9] Hung-Yu Chien; "SASI: A New Ultralightweight RFID Authentication Protocol Providing Strong Authentication and Strong Integrity," *IEEE Transactions on Dependable and Secure Computing,* vol.4, no.4, pp.337-340, Oct.-Dec. 2007.

[10] Pedro Peris-Lopez, Julio Cesar Hernandez-Castro, Juan M.E. Tapiador, and Arturo Ribagorda, “Advances in Ultralightweight Cryptography for Low-Cost RFID Tags: Gossamer Protocol”, *Springer,* pp. 56-68, 2009.

[11] Phan, R.C.-W.; "Cryptanalysis of a New Ultralightweight RFID Authentication Protocol—SASI," *, IEEE Transactions on Dependable and Secure Computing*, vol.6, no.4, pp.316-320, Oct.-Dec. 2009.

[12] Kulseng, L.; Zhen Yu; Yawen Wei; Yong Guan; , "Lightweight Mutual Authentication and Ownership Transfer for RFID Systems," *IEEE INFOCOM, 2010* , pp.1-5, 14-19 March 2010.

[13] Bolotnyy, L.; Robins G.; “Physical Privacy and Security in RFID Systems”, *Auerbach Publications,* 2009.

[14]The NetBeans homepage: http://www.netbeans.org.

[15] The Java homepage: http://www.oracle.com/us/ technologies/java/

[16] The Wireshark network protocol analyzer homepage: www.wireshark.org