How to SQUASH your Data

Adi Shamir

Department of Computer Science and Applied Mathematics
Weizmann Institute of Science

CRYPTO'07 rump session 21/8/2007

What humanity **REALLY** needs at this point in time:

What humanity REALLY needs at this point in time:

- new hash functions!!!

What humanity REALLY needs at this point in time:

- new hash functions!!!

Example: RFID tags

RFID tags have many practical applications:

- UCSB dormitory keys
- warehouse inventory control
- supermarket checkout counters
- public transportation passes
- anti counterfeiting tags
- pet identification
- secure passports
- etc...

The current state of RFID security:

The current state of RFID security:

The major requirements for RFID security protocols:

- small footprint (to make the tag cheap)
- low power consumption (to maximize operating range)
- reasonable speed
- good security

The most important security functionality: Challenge/response authentication of RFID tags

- The reader and tag share a 64-bit secret S

The most important security functionality: Challenge/response authentication of RFID tags

- The reader and tag share a 64-bit secret S
- The reader sends a random 64-bit challenge R

The most important security functionality: Challenge/response authentication of RFID tags

- The reader and tag share a 64-bit secret S
- The reader sends a random 64-bit challenge R
- The tag responds with a 32-bit response H(S,R)

The most important security functionality: Challenge/response authentication of RFID tags

- The reader and tag share a 64-bit secret S
- The reader sends a random 64-bit challenge R
- The tag responds with a 32-bit response H(S,R)
- The reader checks by computing the expected response

Security requirements for the hash function H:

- Should be one-way to protect S
- Need not be collision resistant
- Similar to a MAC, but operating on a single block

Our new SQUASH function: One size fits all

Our new SQUASH function: One size fits all

- We call the new hash function SQUASH which is a squashed form of SQUare-hASH

Our new SQUASH function: One size fits all

- We call the new hash function SQUASH which is a squashed form of SQUare-hASH
- It is provably at least as one-way as Rabin's scheme, which is believed to be an excellent one-way function but completely unsuitable for RFID applications: its inputs are too long, its outputs are too long, it takes too much time, and it requires too much memory.

Our new SQUASH function: One size fits all

- We call the new hash function SQUASH which is a squashed form of SQUare-hASH
- It is provably at least as one-way as Rabin's scheme, which is believed to be an excellent one-way function but completely unsuitable for RFID applications: its inputs are too long, its outputs are too long, it takes too much time, and it requires too much memory.
- SQUASH has an extremely small footprint on RFID tags

Our new SQUASH function: One size fits all

- We call the new hash function SQUASH which is a squashed form of SQUare-hASH
- It is provably at least as one-way as Rabin's scheme, which is believed to be an excellent one-way function but completely unsuitable for RFID applications: its inputs are too long, its outputs are too long, it takes too much time, and it requires too much memory.
- SQUASH has an extremely small footprint on RFID tags
- SQUASH is extremely fast on PC's with arbitrary word sizes

The SQUASH algorithm is very simple:

The SQUASH algorithm is very simple:

- Initialize: Set $\boldsymbol{j} \leftarrow \mathbf{666}$, $\boldsymbol{k} \leftarrow \mathbf{1277}$ and $\boldsymbol{c} \leftarrow \mathbf{0}$

The SQUASH algorithm is very simple:

- Initialize: Set $\boldsymbol{j} \leftarrow \mathbf{666}$, $\boldsymbol{k} \leftarrow \mathbf{1277}$ and $\boldsymbol{c} \leftarrow \mathbf{0}$
- Expand: Seed two 64-bit nonlinear feedback shift registers with $\boldsymbol{S} \oplus \boldsymbol{R}$

The SQUASH algorithm is very simple:

- Initialize: Set $\boldsymbol{j} \leftarrow \mathbf{666}$, $\boldsymbol{k} \leftarrow \mathbf{1277}$ and $\boldsymbol{c} \leftarrow \mathbf{0}$
- Expand: Seed two 64-bit nonlinear feedback shift registers with $\boldsymbol{S} \oplus \boldsymbol{R}$
- Convolve: Compute $\boldsymbol{c} \leftarrow \boldsymbol{c} + \sum_{v=0}^{k-1} \boldsymbol{m_v} * \boldsymbol{m_{j-v(mod\ k)}}$, $\boldsymbol{c} \leftarrow \lfloor \boldsymbol{c}/\mathbf{2} \rfloor$, increment j

The SQUASH algorithm is very simple:

- Initialize: Set $\boldsymbol{j} \leftarrow \mathbf{666}$, $\boldsymbol{k} \leftarrow \mathbf{1277}$ and $\boldsymbol{c} \leftarrow \mathbf{0}$
- Expand: Seed two 64-bit nonlinear feedback shift registers with $\boldsymbol{S} \oplus \boldsymbol{R}$
- Convolve: Compute $\boldsymbol{c} \leftarrow \boldsymbol{c} + \sum_{v=0}^{k-1} \boldsymbol{m_v} * \boldsymbol{m_{j-v(mod\ k)}}$, $\boldsymbol{c} \leftarrow \lfloor \boldsymbol{c}/\mathbf{2} \rfloor$, increment j
- Repeat: Repeat the convolution 48 times, and output the sequence of LSB's of c in the last 32 iterations.

- Thats all, folks!