Hash-based commitment schemes without a computer

Description	A few simple hash algorithms which provide some security
Period	Spring 2016
Languages & Libs	Pseudo
Tags	Encryption, Stack Overflow

An interesting question was posted to crypto.stackexchange.com: "Is there a simple hash function that one can compute without a computer?" Here are three proposed algorithms based on Zobrist hashing, RC4 and A5/1. These should be reasonably secure even against attacks with a calculator, except the one based on Zobrist hashing (but I don't know how to prove or dis-prove this claim). These constructs are especially well suited for commitment schemes.

The first algorithm is based on Zobrist hashing, which is commonly used at chess engines and other similar applications to implement transposition tables. It is based on pre-generating random integers to a look-up table, and then selectively `XOR`ing them together to calculate a hash. This construct has tunable parameters which allow for a trade-off between security and the ease of computation.

For relatively high security it is proposed to use 64 random 80-bit integers. At crypto.stackexchange.com I suggested to use 256-bit integers but it turns out you don't need so many bits to be almost certain to have all of them linearly independent on `GF(2)`.

Based on simulations (see Figure 1 for details) 80×64 bits has approximately $1 : 2^{80-63.8} = 1 : 2^{16.2} \approx 1 : 10^{4.877}$ probability of having non-full rank. A random 64×64 matrix has a non-full rank with a probability of about 71.1%. Each additional bit drops this by 50%. The simplest way to generate this pool of random integers is to flip n coins $64 \cdot 80/n$ times, as you get 1 bit of randomness / coin / toss. This table needs to be published and/or communicated to the 2nd party in advance and can be used multiple times. Numbers are identified as $r_0, r_1, ..., r_{63}$.

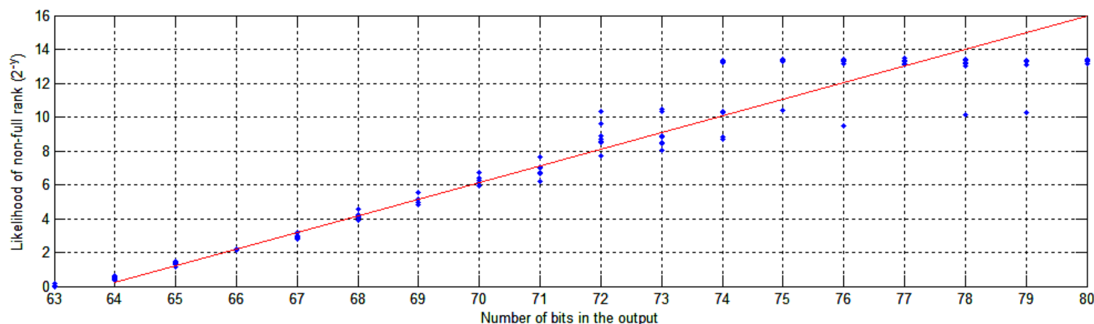

Figure 1: Likelihood of non-full rank approximately halves when an extra bit (row) is added.

Commitment to a yes/no answer and be encoded as the parity of a binary integer $v = 2^{i_0} + 2^{i_1} + ... + 2^{i_n}$ (where $i_{j-1} < i_j$). It is suggested to generate first 63 bits by random and choose the final bit according to the desired commitment. Then calculate the 80-bit "committed value" $c = r_{i_0} \oplus r_{i_1} \oplus ... \oplus r_{i_n}$, which is communicated to the 2nd party.

Calculating the commitment value c from from a m-bit input value v is $O(m)$ operation, but recovering v from c is $O(m^3)$ effort. Thus with sufficiently large m it takes "too much" effort to reverse the process without access to a programmable calculator. However it is trivial to do on any phone or laptop.

An alternative scheme is based on the well-known (and seriously flawed) RC4 stream cipher. On a setting with no access to a computer it should be sufficient to use less than 256×8 bits of state. As little as 32 or 64 integers could be sufficient against an attacker who does not possess a calculator, but I have no clue how to evaluate this.

The algorithm itself is well-described on Wikipedia and its operations are shown in Figure 2. It is a bit of an open question what is the best way of seeding the initial state, how many of the initial bits to skip and how many bits of output is needed for given security criteria. A simple solution is to use the standard key-scheduling algorithm, discard first $3 \cdot n$ output bits (n being the number of internal state numbers) and then keeping the desired amount of output (like 80 or 128 bits) as the committed value. Even with all the flaws of RC4-based hashing, without access to an computer it should be difficult to recover the used key from this algorithm's output.

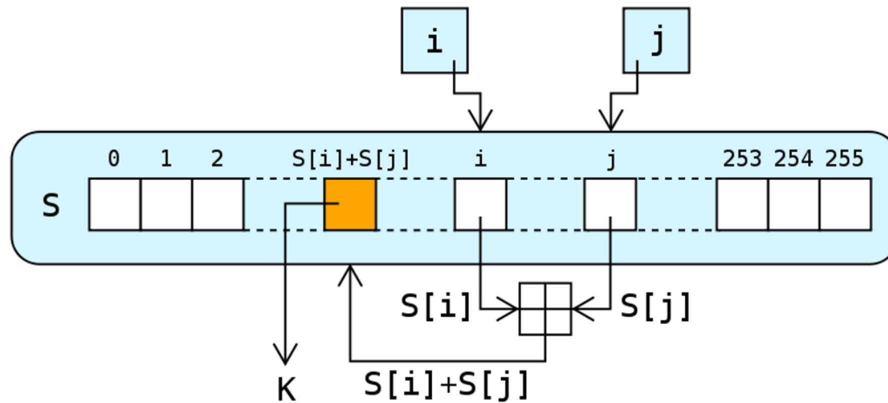

Figure 2: RC4 steps from Wikipedia, just summation and random access to a small chunk of memory.

The final proposed algorithm is similar to RC4: A5/1 which used to encrypt GMS traffic. It has $19 + 22 + 23 = 64$ bits of state, so you'd actually just need 64 coins to represent the state (heads vs. tails). It consist of only simple operations but they are quite verbose to execute manually. Suggested protocol is to commit to an initial state, skip first 64 - 192 bits and publish next 80 bits as the committed value. Again it should be difficult to recover the initial state by only observing this output.

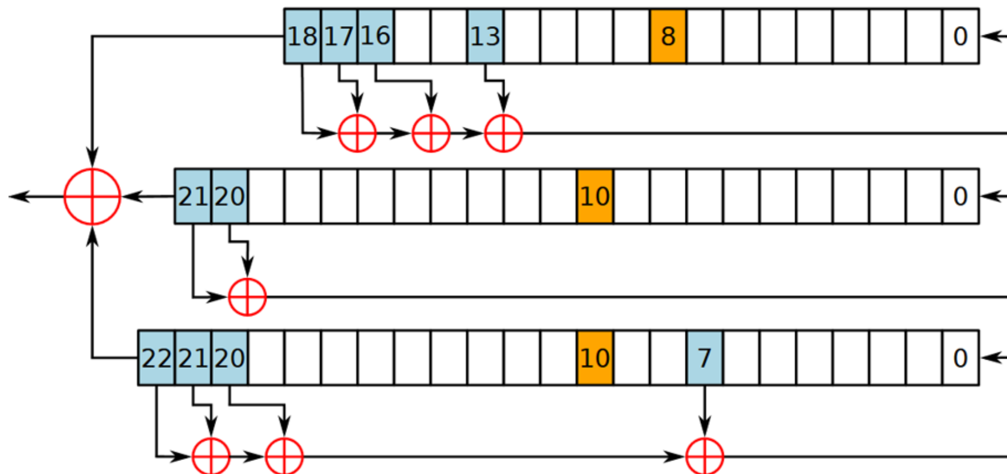

Figure 3: A5/1 steps from Wikipedia, consisting of only simple XOR-operations and look-ups.