Optimizing Honest Majority Threshold Cryptosystems

Ivan Bjerre Damgård, Aarhus University 2020

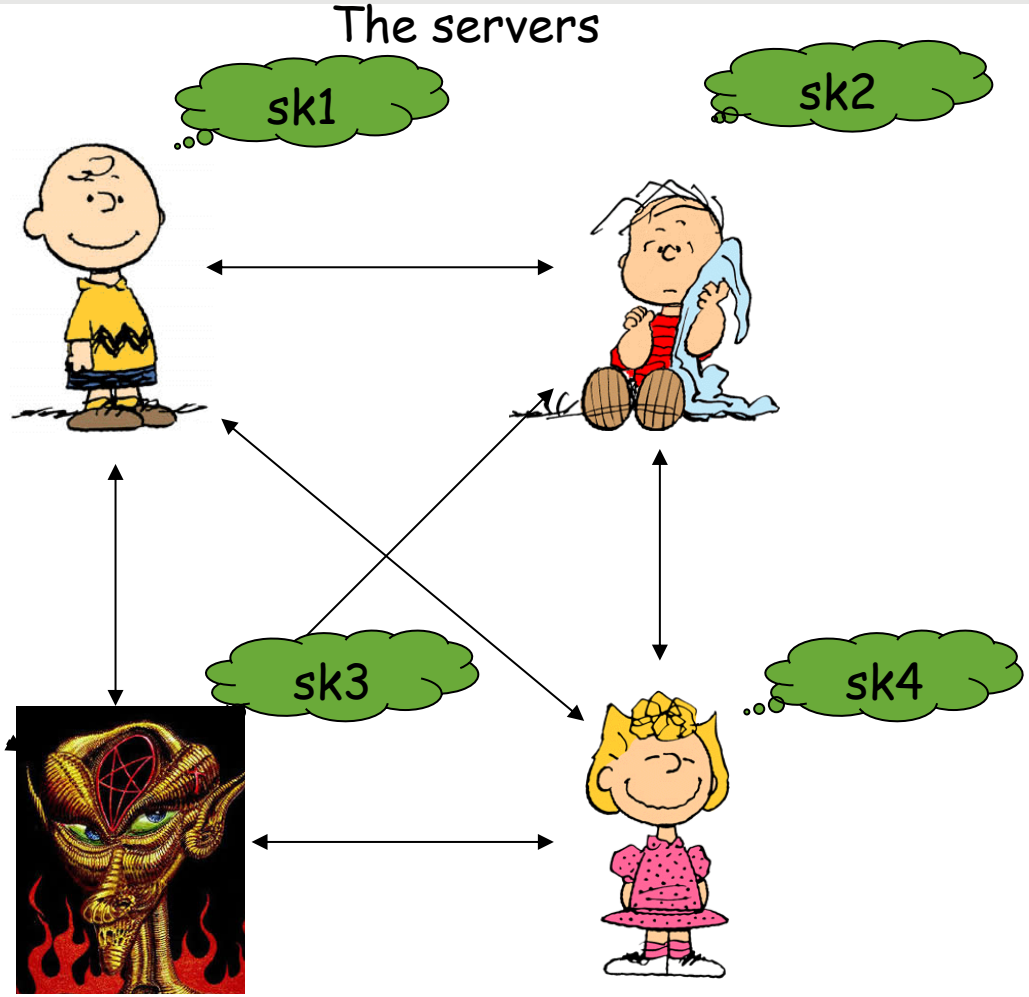
Based on ”Efficient Threshold RSA Signatures with General Moduli and No Extra Assumptions”
By Ivan Damgård and Kasper Dupont, PKC 2005

Threshold Signatures, the setting

The n servers receive from a client a message m to sign. Adv should learn only the signature

Adv

For simplicity, will assume static corruption: Adversary corrupts initially.



Global public signature key pk, Shares sk_i of secret key sk.

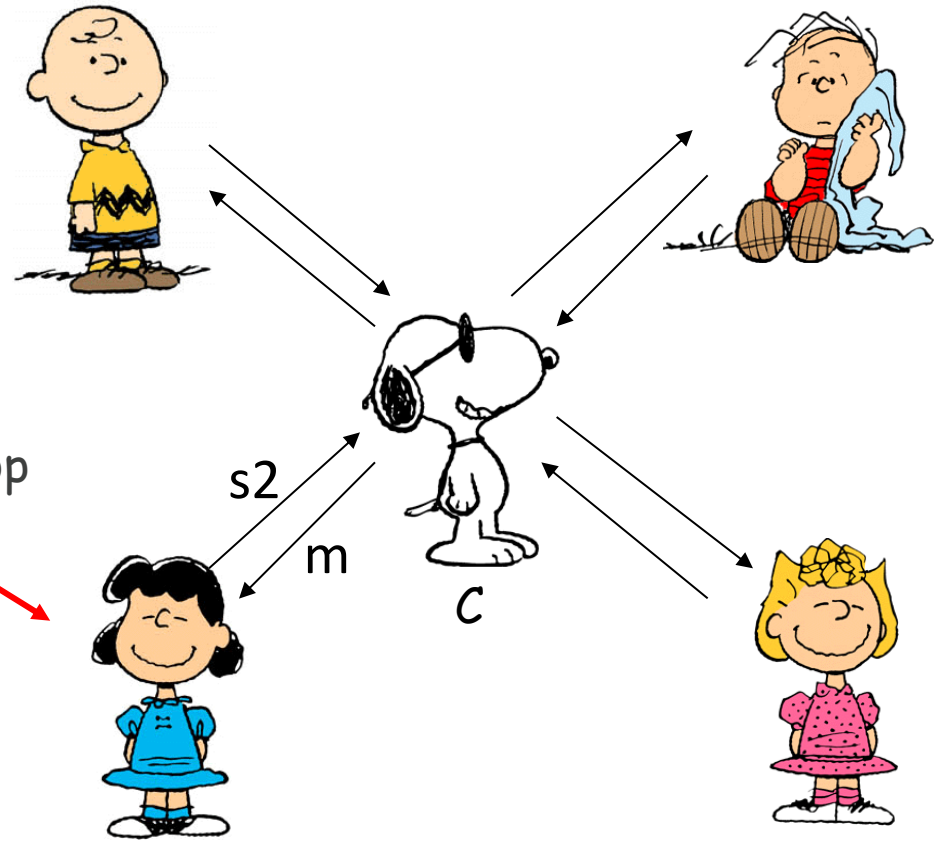
The **Adversary** corrupts a minority actively

A weaker protocol as starting point

Adv

Passive and Fail-stop

The adversary can learn a minority of the sk_i's and all signature shares, and can make corrupted players crash, still learns only signature



Assume protocol is ”non-interactive”: a client can broadcast m to all servers, and servers return signature shares s1,..,sn

=> Client can now compute the signature, using a CombineShares function.

Notation

H a set of indices. $s_H = \{s_i \mid i \text{ in } H\}$.

Our assumption is that if we are given a set of correct signature shares s_H computed from message m, and $|H| \geq n/2$, then

$S_{sk}(m) = \text{CombineShares}(s_H)$

Many examples of this set-up: RSA, Schnorr, DSA, etc.

Specifically, for RSA, ”semihonest with crash” security can be achieved with no special requirements on the modulus and no computational assumptions other than RSA.

How to get malicious security?

The reflex reaction:

Servers should prove in ZK that their signature schares are correct.

This works because at least n/2 correct shares will survive, and all incorrect shares are rejected.

However, main take-home message from this talk:

This is overkill!

An observation

The semi-honest/crash secure protocol is in some sense already maliciously secure. The client can compute the signature as follows:

Given the set of all signature shares, for all subsets H with |H|=n/2,

Compute s= CombineShares(s_H).

If V_{pk}(s,m)= accept, output s and stop.

This works because there is a set H with only honest players and CombineShares is assumed to work for that set.

But of course it scales terribly..

Another observation

We could make the previous algorithm efficient, if we could get rid of enough incorrect shares.

Say we could detect most of the incorrect shares and kick them out. Such that we now have n’ shares left, and h of these are correct, where n’ is only marginally larger than h.

Now, most of the subsets H we try in the algorithm will work, and we will finish much faster.

More precisely

Say we have t corrupted signature shares and t+1 correct ones (worst case, assuming honest majority).

Assume we have a test that accepts a correct share and rejects an incorrect one, except with probability p. We run this test on all input shares and then do the algorithm from before.

Lemma. Let c be any constant such that c>2. if $p < 1/ct^2$, then the expected number of subsets we have to try is O(1).

How to do the test

A simple way to implement the test we assumed is to do a ZK proof with non-negligible soundness error p.

An Example: RSA.

Modulus n, share of secret key sk_i, for i=1..n.

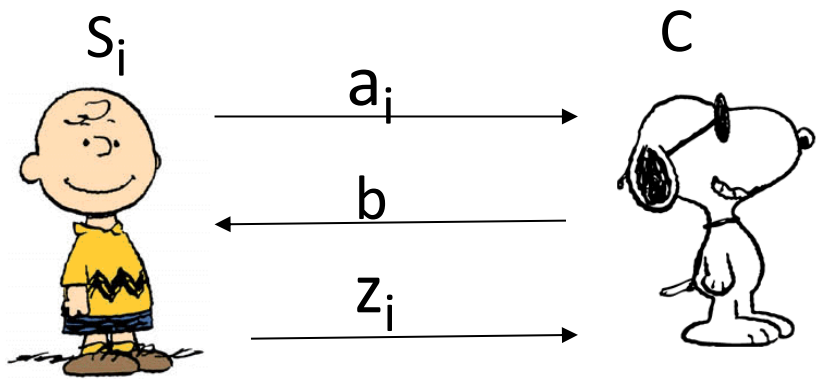
Choose fixed element g of maximal order.

In key generation, compute $h_i = g^{ski} \bmod n$.

Now, in the simplest case, we would do the standard Sigma protocol with 1-bit challenge:

The Sigma protocol

Public: n, g and $h_i = g^{ski} \bmod n$. S_i knows secret key share sk_i.
S_i computes signature share $s_i = m^{ski} \bmod n$



Client chooses a 1-bit challenge b.

In the end, accepts or rejects. If s_i incorrect, will accept with probability at most ½, or Si could be used to factor n.

This works for any modulus, and requires no extra assumptions for soundness, unlike other constructions.

Can repeat in parallel $\log(2t^2)$ times to get soundness error as required in the lemma.

Can also get the same in 1 repetition and a $\log(2t^2)$ –bit challenge under mild restrictons on n.

Caveat: cannot use Fiat-Shamir to make this non-interactive! However…

Optimistic Protocol to avoid interaction

1. C broadcasts m to servers.
2. Each Si sends s_i and a_i (first message in proof). The randomness for this generated by a PRF using a secret key and message m as input (plus perhaps a session id).
3. C tries to compute signature from a t+1 subset. If this works, we are done. Else C sends m, s_i, a_i and a challenge b to the servers.
4. Each Si regenerates its randomness and checks C sent correct data. If so, return the answer z_i in the proof.
5. C can now filter out most bad signature shares and compute signature.

No interaction if servers behave or crash.

Servers need not keep state.

Generalizations

- Can potentially use this idea for any signature scheme because we can verify the output (but the advantage is not always as great as for RSA)
- Works also for RSA decryption, and potentially for any other scheme where we can recognize the correct plaintext when we see it.
- Assume that the number t of servers that can be corrupted is very small compared to the number n of servers – or is almost equal to n. Then, there are not too many t+1 subsets. And the approach works with no ZK at all.
- Notice:the secret sharing scheme in the original paper has a bug, but this is orthogonal to what we discuss here.

Thanks!