Simplification of Controlled PUF primitives

Boris Škorić and Marc X. Makkes

Physical Unclonable Functions (PUFs) are physical objects that are unique, practically unclonable and that behave like a random function when subjected to a challenge. Their use has been proposed for authentication tokens and anti-counterfeiting. A Controlled PUF (CPUF) consists of a PUF and a control layer that restricts a user's access to the PUF input and output. CPUFs can be used for secure key storage, authentication, certified execution of programs, and certified measurements. In this paper we modify a number of protocols involving CPUFs in order to improve their security. Our modifications mainly consist of encryption of a larger portion of the message traffic, and additional restrictions on the CPUF accessibility. We simplify the description of CPUF protocols by using flowchart notation. Furthermore we explicitly show how the helper data for the PUFs is handled.

The concept of a CPUF was introduced by Gassend et al. in 2002. A CPUF is a combination of a PUF and a control layer in which the PUF is inseparably embedded. The control layer completely shields off the PUF inputs and outputs from the outside world. Any communication with the PUF has to occur through the control layer electronics. Any attempt to force the components apart will damage the PUF. A CPUF has considerably stronger security than a bare, unprotected PUF, since attackers cannot probe and query the PUF at will. In effect, the CPUF is a sort of trusted computing environment. A way was presented by Gassend et al. to employ this trusted environment for the purpose of outsourcing computations. The idea is roughly as follows. First, Challenge-Response Pairs (CRPs) of the PUF are handed to users in a secure way. Everybody can remotely run programs on the CPUF control layer. There is a special Application Programming Interface (API) for accessing the PUF. With the help of this API a user can instruct the CPUF to generate a 'proof' of the correct execution of the outsourced program. This proof can be thought of as a MAC over the executed program and the program output, using the PUF response as the MAC key. If the user has a valid CRP, he can verify the MAC. This procedure is referred to as 'certified execution'. The proof is verifiable only by the user who sends the task to the CPUF. This was later generalized to a proof ('E-proof') that can be verified by third parties as well.

In this paper we propose a modification of the main CPUF security primitives. These modifications improve the overall security by putting additional restrictions on access to the CPUF and by encrypting more of the exchanged messages. We represent the protocols in a different way from Gassend et al., namely in the form of flowcharts, which improves the comprehensibility of the protocols and of their security properties. In the API formulation, hashes of API programs play an important role in the security primitives. In some cases, a function call involves a hash of a piece of the program containing the function call. We feel that such a formulation is needlessly complicated. Especially the self-referential nature of the

Dagstuhl Seminar Proceedings 09282
Foundations for Forgery-Resilient Cryptographic Hardware
http://drops.dagstuhl.de/opus/volltexte/2010/2404

program hashes is confusing. In our flowchart notation, each security primitive corresponds to a 'mode' of the CPUF, in which the control layer has a certain fixed input/output behaviour. A user can instruct a CPUF in which mode to operate, but cannot change the CPUF's sequence of actions in that mode. For each mode we present a flowchart. There are no hashes of control layer programs; the security clearly derives from the secrecy of the challenge-response pairs. Avoiding the program hashes allows for more efficient implementation.

In contrast to Gassend et al., we do not allow just anybody to outsource computations to the CPUF, but we first demand that a user establishes a secure channel with the control layer, based on a shared CRP. Any further communication has to take place through this channel. The advantage of this approach is twofold: (i) it provides more data confidentiality, e.g. the outsourced job and the results are not revealed to eavesdroppers, and (ii) it restricts the opportunities for attacks. Finally, we explicitly show how the helper data is handled; this makes no essential difference with respect to the prior literature but completes the data flow overview.

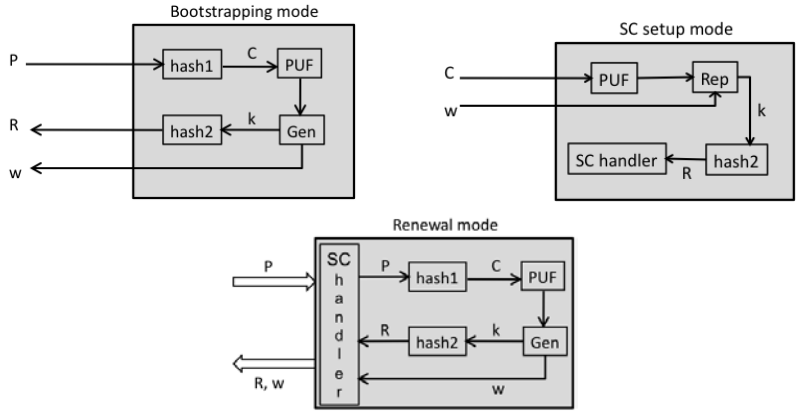

Fig. 1. *Flowcharts for bootstrapping, Secure Channel setup and renewal. P =pre-challenge; C = PUF challenge; w =helper data; k =key; R = response.*

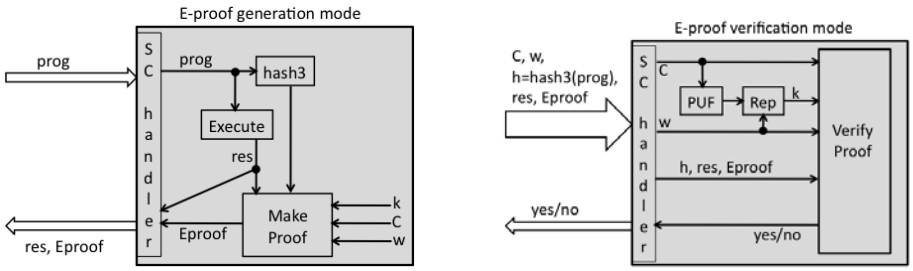

Fig. 2. *The E-proof generation and verification are run through a Secure Channel (SC). The SC-key in use is the hash of the secret key k; this k never leaves the CPUF. The key k is used by the* `MakeProof` *function to certify the program hash, the result of the computation, and the SC setup parameters C, w.*