Direct Anonymous Attestation Explained

Jan Camenisch
IBM Research
jca@zurich.ibm.com

July 25, 2007

1 Introduction

Assume that the user a trusted computing platform communicates with a verifier who wants to be assured that the user indeed uses a platform that can be trusted. trusted hardware module, This problem is called remote attestation and discussed in detail in Chapter ??. As described there, the problem in the end boils down to the problem that a trusted platform module (TPM) needs to prove that the attestation identity key (AIK) it has generated was indeed generated by an authentic trusted platform module. In principle, the TPM could use its endorsement key (EK) together with its certificate on this key to authenticate the AIK. However, the user wants her privacy protected and therefore requires that the verifier only learns that she uses a TPM but not which particular one – thus such a solution does not work as all her transactions would become linkable to each other by the EK.

Another solution to the problem could be using any standard public key authentication scheme (or signature scheme): One would generate a secret/public key pair, and then embed the secret key into each TPM. The verifier and the TPM would then run the authentication protocol. Because all TPMs use the same key, they are indistinguishable. However, this approach would never work in practice: as soon as one hardware module (TPM) gets compromised and the secret key extracted and published, verifiers can no longer distinguish between real TPMs and fake ones. Therefore, detection of rogue TPMs needs to be a further requirement.

The solution first developed by TCG uses a trusted third party, the so-called privacy certification authority (Privacy CA), and works as follows [25]. Each TPM generates an RSA key pair called an Endorsement Key (EK).

and computes

$$\tilde{U} := R_0^{r_{f_0}} R_1^{r_{f_1}} S^{r_v} \bmod n \ ,$$
$$\tilde{r}_f := r_{f_0} + r_{f_1} 2^{\ell_f} \bmod \rho \ , \text{ and}$$
$$\tilde{N}_V := \zeta^{\tilde{r}_f} \bmod \Gamma \ .$$

The TPM sends $\tilde{T}_{1t}$ and $\tilde{N}_V$ to the host.

iii. The host picks random integers

$$r_e \in_R \{0,1\}^{\ell'_e + \ell_\varnothing + \ell_{\mathcal{H}}} \quad \text{and} \quad r_w \in_R \{0,1\}^{\ell_e + \ell_n + \ell_\varnothing + \ell_{\mathcal{H}}}$$

and computes

$$\tilde{A}' := \tilde{U} A'^{r_e} S^{r_w} \bmod n$$

(b) i. Host receives the nonce n_v from the verifier and computes

$$c_h := H((n \| g \| g' \| h \| R_0 \| R_1 \| S \| Z \| \gamma \| \Gamma \| \rho) \| \zeta \| A' \| N_V \| \tilde{A} \| \tilde{N}_V) \| n_v) \ .$$

and sends c_h to the TPM.

ii. The TPM chooses a random $n_t \in \{0,1\}^{\ell_\varnothing}$ and computes

$$c := H(H(c_h \| n_t) \| b \| m) \ .$$

and sends c, n_t to the host.

(c) i. The TPM computes (over the integers)

$$s_\nu := r_v + c \cdot v \ , \quad s_{\varphi_0} := r_{f_0} + c \cdot f_0 \ , \quad \text{and}$$
$$s_{\varphi_1} := r_{f_1} + c \cdot f_1$$

and sends (s_v, s_{f_0}, s_{f_1}) to the host.

ii. The host computes (over the integers)

$$s_\varepsilon := r_e + c \cdot (e - 2^{\ell_e - 1}) \quad \text{and} \quad s_{\tilde{\nu}} := s_\nu + r_w + c \cdot (e \cdot w) \ .$$

4. The host sends the signature

$$\sigma := (\zeta, A', N_V, c, n_t, (s_{\tilde{\nu}}, s_{\varphi_0}, s_{\varphi_1}, s_\varepsilon)) \tag{3}$$

to the verifier.

5. The verifier first checks whether the values contained in σ are a valid signature of knowledge

$$SPK\{(\varepsilon, \tilde{\nu}, \varphi_0, \varphi_1) :$$
$$ZA'^{-2^{\ell_e-1}} \equiv \pm A'^{\varepsilon} S^{\tilde{\nu}} R_0^{\varphi_0} R_1^{\varphi_1} \pmod{n} \wedge$$
$$N_V \equiv \zeta^{\varphi_0+\varphi_1 2^{\ell_f}} \pmod{\Gamma} \wedge$$
$$\varphi_0, \varphi_1 \in \{0,1\}^{\ell_f+\ell_\varnothing+\ell_\mathcal{H}+2} \wedge \varepsilon \in \pm\{0,1\}^{\ell'_e+\ell_\varnothing+\ell_\mathcal{H}+2}\}(b\|m) \ .$$

That is, the verifier

(a) computes

$$\hat{A}' := (ZA'^{-2^{\ell_e-1}})^{c} R_0^{s_{\varphi_0}} R_1^{s_{\varphi_1}} A'^{s_\varepsilon} S^{s_{\tilde{\nu}}} \bmod n \qquad \text{and}$$
$$\hat{N}_V := N_V^{c} \zeta_V^{s_{\varphi_0}+2^{\ell_f} s_{\varphi_1}}$$

and verifies whether

$$c \stackrel{?}{=} H(H(H((n\|g\|g'\|h\|R_0\|R_1\|S\|Z\|\gamma\|\Gamma\|\rho)\|\zeta\| A'\|N_V\|\hat{A}'\|\hat{N}_V)\|n_v)\|n_t)\|b\|m) \ .$$

and

$$s_{\varphi_0}, s_{\varphi_0} \stackrel{?}{\in} \pm\{0,1\}^{\ell_f+\ell_\varnothing+\ell_\mathcal{H}+1} \qquad \text{and} \qquad s_\varepsilon \stackrel{?}{\in} \pm\{0,1\}^{\ell'_e+\ell_\varnothing+\ell_\mathcal{H}+1}$$

all holds.

6. If ζ was derived from the verifier's base name, the verifier also checks whether $\zeta \stackrel{?}{\equiv} (H_\Gamma(1\|\mathtt{bsn}_V))^{(\Gamma-1)/\rho} \pmod{\Gamma}$.

7. Finally, for all (f_0, f_1) on the rogue list, the verifier checks whether $N_V \stackrel{?}{\not\equiv} (\zeta^{f_0+f_1 2^{\ell_f}}) \pmod{\Gamma}$.

4.7 Security Discussion

A full proof of security (and a formal model of DAA) is outside the scope of this chapter. Let us discuss here informally why the protocols described above meet our goals.

The informal properties that we wanted to achieve are unforgeability, anonymity, and rogue tagging (cf. 4.1). Unforgeability requires that a host