ICALP 2005, Lisboa, Portugal

Designated Verifier Signatures: Attacks, New Definitions and Constructions

Helger Lipmaa

Cybernetica AS and University of Tartu, Estonia

Guilin Wang and Feng Bao

Institute of Infocomm Research, Singapore

Outline

- Motivation for DVS
- Attacks on Some Previous Constructions
- New Security Notions
- Our Own Construction
- Conclusion

Outline

- Motivation for DVS
- Attacks on Some Previous Constructions
- New Security Notions
- Our Own Construction
- Conclusion

Motivation

I w4nt 2 read s0me b00k.
But I h4ve 2 b a subscr1b3r!
Th1s 1s ok, I c4n s1gn my request
But 1 do not w4nt Sl1ck to show
the s1gnatur3 2 oth3rs!

Motivation

I w4nt 2 read s0me b00k.
But I h4ve 2 b a subscr1b3r!
Th1s 1s ok, I c4n s1gn my request
But 1 do not w4nt Sl1ck to show
the s1gnatur3 2 oth3rs!

My fr1end Markus sa1d I can
us3 des1nated ver1f1er s1gnatures!
S1nce Desmond can s1mulate such
s1gnatures, the s1gnatures are
non–transferable.

Hej! I am Markus.

More applications?

- Service providing/Privacy-preserving data-mining:
  - ⋆ Desmond knows Signy is a loyal customer; Signy gets bonus
  - ⋆ Desmond can add information about Signy in the database and process it later
  - ⋆ Desmond can't prove to anybody else that the database is correct but he trusts himself!
- E-voting: Signy is a voter, Desmond is a tallier. Desmond knows that Signy voted but cannot prove it to anybody else.
- Etc etc etc

Thus spake Markus to Signy:

Public key $y_S = g^{x_S}$

Public key $y_D = g^{x_D}$

Thus spake Markus to Signy:

Signy does

Public key $y_S = g^{x_S}$

Generate $s \leftarrow m^{x_S}$

Public key $y_D = g^{x_D}$

Thus spake Markus to Signy:

Signy does

Public key $y_S = g^{x_S}$

Generate $s \leftarrow m^{x_S}$

Generate random $w, t, r \leftarrow \mathbb{Z}_q$

Public key $y_D = g^{x_D}$

Thus spake Markus to Signy:

Signy does

Public key $y_S = g^{x_S}$

Generate $s \leftarrow m^{x_S}$

Generate random $w, t, r \leftarrow \mathbb{Z}_q$

Set $h \leftarrow H(g^w y_D^t, g^r, m^r)$

Public key $y_D = g^{x_D}$

Thus spake Markus to Signy:

Signy does

Public key $y_S = g^{x_S}$

Public key $y_D = g^{x_D}$

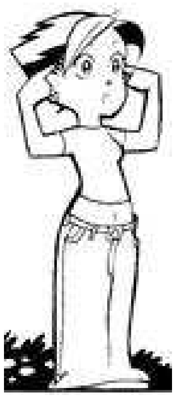
Generate $s \leftarrow m^{x_S}$
Generate random $w, t, r \leftarrow \mathbb{Z}_q$
Set $h \leftarrow H(g^w y_D^t, g^r, m^r)$
Set $z \leftarrow r + (h + w)x_S$

Signature $\sigma = (s; w, t, h, z)$ →

Thus spake Markus to Signy:

Signy does

Public key $y_S = g^{x_S}$



Generate $s \leftarrow m^{x_S}$

Generate random $w, t, r \leftarrow \mathbb{Z}_q$

Set $h \leftarrow H(g^w y_D^t, g^r, m^r)$

Set $z \leftarrow r + (h + w) x_S$

Public key $y_D = g^{x_D}$

Signature $\sigma = (s; w, t, h, z)$

Verify that $h = H(g^w y_D^t, \underbrace{g^z y_S^{-(h+w)}}_{g^{z-(h+w)x_S} = g^r}, \underbrace{m^z s^{-(h+w)}}_{m^{z-(h+w)x_S} = m^r})$

Thus spake Markus to Desmond:

Desmond does

Public key $y_S = g^{x_S}$

Public key $y_D = g^{x_D}$

Choose *any* s

Generate random $z, \alpha, \beta \leftarrow \mathbb{Z}_q$

Set $h \leftarrow H(g^{\alpha}, g^{z} y_S^{-\beta}, m^{z} s^{-\beta})$

Set $w \leftarrow \beta - h$, $t \leftarrow (\alpha - w) x_D^{-1}$

Signature $\sigma = (s; w, t, h, z)$

Verify that $h = H(\underbrace{g^{w} y_D^{t}}_{g^{w+t\cdot x_D}}, \underbrace{g^{z} y_S^{-(h+w)}}_{g^{z} y_S^{-\beta}}, \underbrace{m^{z} s^{-(h+w)}}_{m^{z} s^{-\beta}})$

Das ist ja Korrekt!

Thus spake Markus to both:

- If Signy signs: $s = m^{x_S}$, thus (g, y_S, m, s) is a DDH tuple
  - $\star$ $(g, y_S, m, s) = (g, g^a, g^b, g^{ab})$ for some a, b
- Signy proves in NIZK that (g, y_S, m, s) is a DDH tuple
- If Desmond simulates: *any* $\overline{s}$; since DL is hard, $(g, y_S, m, \overline{s})$ is not a DDH tuple w.h.p. $1 - \frac{1}{q}$
  - $\star$ $c = g^w y_D^t$ for which Desmond knows the trapdoor x_D
  - $\star$ Desmond can simulates the proof by using the trapdoor *for any* $\overline{s} \in \mathbb{Z}_p$
- Signy can disavow, w.h.p. $1 - \frac{1}{q}$, by proving that $\overline{s} \neq m^{x_S}$

Thus spake Markus to both:

- To generate a valid $\sigma \leftarrow (s; w, t, h, z)$ you must know either x_S or x_D

- Thus Desmond knows σ was generated by Signy

  - ⋆ Since Desmond did not generate it himself

- Any third party doesn't know whether σ was generated by Signy or Desmond

And Signy was very happy and Desmond coverted in snow.

But Desmond met Guilin and Guilin spake to him:

But Desmond met Guilin and Guilin spake to him:

Public key $y_S = g^{x_S}$

Generate random $w, t, r \neq \bar{r} \leftarrow \mathbb{Z}_q$
Set $h \leftarrow H(g^w y_D^t, g^r, m^{\bar{r}})$
Set $z \leftarrow r + (h + w) x_S$
Set $\bar{s} \leftarrow m^{x_S} \cdot m^{(r-\bar{r})/(h+w)}$

Public key $y_D = g^{x_D}$

Signature $\sigma = (\bar{s}; w, t, h, z)$

Signy can also do this!

Verify that $h = H(g^w y_D^t, \underbrace{g^z y_S^{-(h+w)}}_{g^{z-(r-\bar{r})}=g^r}, \underbrace{m^z (\bar{s})^{-(h+w)}}_{m^{z-(h+w)x_S-(r-\bar{r})}=m^{\bar{r}}})$

But Desmond met Guilin and Guilin spake to him:

- Verification succeeds, thus Desmond accepts it as Signy's signature
- However, since $\overline{s} \neq m^{xs}$, Signy can later disavow it!

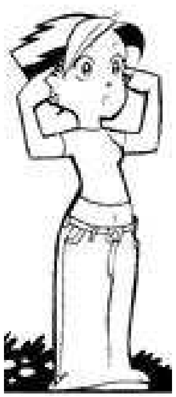
And Desmond was not so happy anymore.

Quick fix:

Signy does

Public key $y_S = g^{x_S}$

Public key $y_D = g^{x_D}$



Generate $s \leftarrow m^{x_S}$

Generate random $w, t, r \leftarrow \mathbb{Z}_q$

Set $h \leftarrow H(g^w y_D^t, g^r, m^r, \underline{\underline{\mathsf{pk}_S}}, \underline{\underline{\mathsf{pk}_D}}, \underline{\underline{s}})$

Set $z \leftarrow r + (h + w) x_S$

Signature $\sigma = (s; w, t, h, z)$

Verify that $h = H(g^w y_D^t, g^z y_S^{-(h+w)}, m^z s^{-(h+w)}, \mathsf{pk}_S, \mathsf{pk}_D, s)$

Then, Signy met some other people

- Steinfeld, Bull, Wang and Pieprzyk said: use a bilinear pairing $\langle \cdot, \cdot \rangle$
  - $\star$ $\langle b^a, d^c \rangle = \langle b, d \rangle^{ac}$ with natural hardness assumptions
- Signy signs m: $s = \langle m^{x_S}, y_D \rangle = \langle m, g \rangle^{x_S x_D}$
- Desmond simulates: $\overline{s} = \langle m^{x_D}, y_S \rangle = \langle m, g \rangle^{x_S x_D}$
- Verification by Desmond: $\langle m, y_S^{x_D} \rangle = s$?

And Signy was happy again and kissed Pieprzyk.

However, Desmond met Guilin again

- Signy signs m: $s = \langle m^{x_S}, y_D \rangle = \langle m, g \rangle^{x_S x_D} = \langle m, g^{x_S x_D} \rangle$

However, Desmond met Guilin again

- Signy signs m: $s = \langle m^{x_S}, y_D \rangle = \langle m, g \rangle^{x_S x_D} = \langle m, g^{x_S x_D} \rangle$

Guilin spake to Desmond:

- Signy can compute $y_{SD} := g^{x_S x_D}$ and publish it

- Then anybody can sign m as $s = \langle m, y_{SD} \rangle = \langle m, g \rangle^{x_S x_D}$

- Thus Signy can delegate her subscription to your library, without revealing her public key

And Desmond wanted to cry.

And so forth and so forth

- Signy and Desmond met many wise men who proposed better and better designated verifier signature schemes.

- However, Guilin broke them all!

- Sad story, eh?

- Signy even thought about never reading a book again!

What went wrong?

- [JSI1996]: disavowability claimed but does not exist
- [SBWP2003] and some other schemes were delegatable

What should we do?

$\Rightarrow$ Propose a DVS scheme that is *unforgeable*

- $\star$ Use as tight reductions as possible
- $\star$ . . . and as weak trust model as possible

$\Rightarrow$ Eliminate disavowal or make it “secure”

- *Non-delegatability* was never considered before

$\Rightarrow$ Define non-delegatability and propose a non-delegatable scheme

Unforgeability: Definition

Consider the next game:

- Choose random key pairs for Signy and Desmond
- Give the Forger both public keys, an oracle access to Signy's signing algorithm, Desmond's simulation algorithm and the hash function
- Forger returns a message m and a signature σ

Forger is **successful** if verification on (m, σ) succeeds and he never asked a sign/simul query on m that returned σ

Scheme is $(\tau, q_h, q_s, \varepsilon)$**-unforgeable** $\iff$ no (τ, q_h, q_s)-forger has success probability $> \varepsilon$

Forger runs in time τ, does q_h queries to hash function and q_s queries to either signing or simulation algorithm

Non-Transferability: Definition

- A scheme is **perfectly** non-transferable if signatures generated by Signy and Desmond come from the same distribution.
  - ⋆ Perfectly non-transferable schemes *cannot* have disavowal protocols!
  - ⋆ As we showed, JSI is perfectly non-transferable!

- A scheme is **computationally** non-transferable if signatures generated by Signy and Desmond come from distributions that are computationally indistinguishable.
  - ⋆ Computationally non-transferable schemes *may* have a trapdoor that can be used for constructing disavowal protocols

Non-Delegatability: Definition

Briefly: A DVS signature is a non-interactive proof of knowledge of either of the secret keys.

Requirement: if Forger produces valid signatures with probability $> \kappa$ then he knows either the secret key of Signy or the secret key of Desmond

We require there exists a knowledge extractor such that

- **If** a Forger produces a valid signature σ on m w.p. $\varepsilon > \kappa$
  then knowledge extractor, given m and oracle access to Forger on input m, produces one of the two secret keys in time $\frac{\tau}{\varepsilon - \kappa}$.

Then the scheme is (τ, κ)-non-delegatable.

Outline

- Motivation for DVS
- Attacks on Some Previous Constructions
- New Security Notions
- Our Own Construction
- Conclusion

Underlying Idea of Our Scheme

- If Signy signs: she proves that her public key $(g_1, g_2, y_{1S} = g_1^{x_S}, y_{2S} = g_2^{x_S})$ is a DDH tuple.

- We again employ $c = g^w y_D^t$ (trapdoor commitment) for which Desmond knows the trapdoor x_D, thus the proof is designated-verifier.

- Desmond simulates this proof by using the trapdoor information

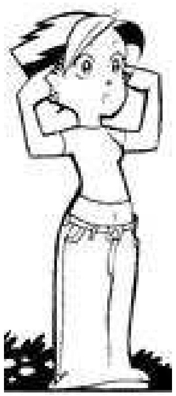
- Signy cannot disavow since there is perfect non-transferability

(Merrily marrying Katz-Wang conventional signature scheme + JSI96 DVS)

And Thus We Spake to Signy:

Signy does

Public key $(y_{1S} = g_1^{x_S}, y_{2S} = g_2^{x_S})$



Generate random $w, t, r \leftarrow \mathbb{Z}_q$

Set $h \leftarrow H(\mathsf{pk}_S, \mathsf{pk}_D, g_1^w y_{1D}^t, g_1^r, g_2^r, m)$

Set $z \leftarrow r + (h + w) x_S$

Public key $(y_{1D} = g_1^{x_D}, y_{2D} = g_2^{x_D})$

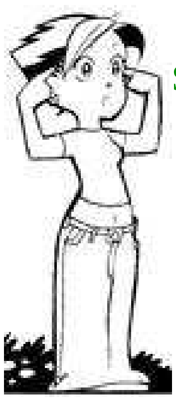
Signature $\sigma = (w, t, h, z)$

Verify that $h = H(\mathsf{pk}_S, \mathsf{pk}_D, g_1^w y_{1D}^t, g_1^z y_{1S}^{-(h+w)}, g_2^z y_{2S}^{-(h+w)}, m)$

And Thus We Spake to Desmond:

Desmond does

Public key $(y_{1S} = g_1^{x_S}, y_{2S} = g_2^{x_S})$



Generate random $z, \alpha, \beta \leftarrow \mathbb{Z}_q$

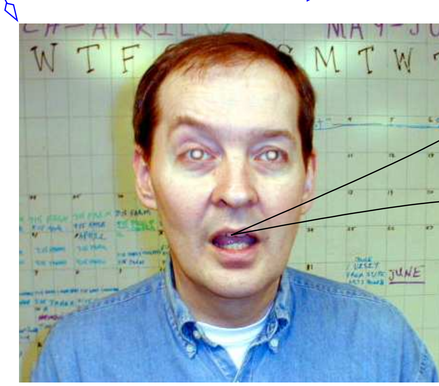
Set $h \leftarrow H(\mathsf{pk}_S, \mathsf{pk}_D, g_1^{\alpha}, g_1^{z} y_{1S}^{-\beta}, g_2^{z} y_{2S}^{-\beta}, m)$

Set $w \leftarrow \beta - h$, $t \leftarrow (\alpha - w) x_D^{-1}$

Signature $\sigma = (w, t, h, z)$

Public key $(y_{1D} = g_1^{x_D}, y_{2D} = g_2^{x_D})$

Verify that $h = H(\mathsf{pk}_S, \mathsf{pk}_D, g_1^{w} y_{1D}^{t}, g_1^{z} y_{1S}^{-(h+w)}, g_2^{z} y_{2S}^{-(h+w)}, m)$



Das ist ja Korrekt!

Properties of The New Scheme

- Twice longer public keys than in JSI — enables to get tight unforgeability reductions
  - ⋆ In *non-programmable random oracle model*
  - ⋆ As in Katz-Wang, unforgeability proof does not use proof of knowledge/forking lemma
- Perfect non-transferability, thus **no disavowal**
  - ⋆ Orthogonal to the security requirements of an DVS scheme
- Non-delegatability: proven, but the reduction is not tight
  - ⋆ Proof of knowledge

Unforgeability

Theorem. Let G, $|G| = q$ be a (τ', ε')-DDH group. The proposed scheme is $(\tau, q_h, q_s, \varepsilon)$-unforgeable in the non-programmable random oracle model with $\tau \leq \tau' - (3.2q_s + 5.6)t_{\text{exp}}$ and $\varepsilon \geq \varepsilon' + q_s q_h q^{-2} + q^{-1} + q_h q^{-2}$.

Proof sketch: Adversary A has to solve DDH on input $(g_1, g_2, y_{1D}, y_{2D})$. Set this to Desmond's public key, and set Signy's public key to be equal to a random DDH tuple (for which A knows the corresponding secret key). Give A an oracle access to Forger. Answer all hash queries truthfully (but store them). Answer all signing and simulation queries by following Signy's algorithm. (Possible since A knows Signy's secret key.) A works in time and with success probability, claimed above.

Note: This is a tight reduction. In practice it means that whenever you can forge a signature—e.g., 2^{-80}—, you can w.h.p. solve DDH in comparable time.

Unforgeability

Theorem. Let G, $|G| = q$ be a (τ', ε')-DDH group. The proposed scheme is $(\tau, q_h, q_s, \varepsilon)$-unforgeable in the non-programmable random oracle model with $\tau \leq \tau' - (3.4 q_s + 5.6) t_{\text{exp}}$ and $\varepsilon \geq \varepsilon' + q_s q_h q^{-2} + q^{-1} + q_h q^{-2}$.

Proof sketch: Adversary A has to solve DDH on input $(g_1, g_2, y_{1S}, y_{2S})$. Set this to Signy's public key, and set Desmond's public key to be equal to a random DDH tuple (for which A knows the corresponding secret key). Give A an oracle access to Forger. Answer all hash queries truthfully (but store them). Answer all signing and simulation queries by following Desmond's algorithm. (Possible since A knows Desmond's secret key.) A works in time and with success probability, claimed above.

Note: Proof in proceedings is faulty. (Change the roles of S and D!)

Delegatability

Theorem. Let $\kappa \geq 1/q$. Assume that for some message m, Forger can produce signature in time τ' and with probability $\varepsilon \geq \kappa$. Then there exists a knowledge extractor that on input a valid signature σ and on black-box oracle access to Forger (with an internal state compatible with σ) can produce one of the two secret keys in expected time $\tau \leq 56\tau'/\kappa$.

Note: This is an imprecise reduction. For example, if Forger has advantage 2^{-30} then Knowledge Extractor works in time $2^{36} \cdot \tau'$, with probability 1.

Conclusions

- And Desmond was happy since only valid subscribers were able to borrow the books.
  - ⋆ And these subscribers could not delegate their subscriptions!
- And Signy was happy since Desmond could not prove that she borrowed these books.

Any questions?

Note: version with corrected proof upcoming