Short Signatures from Bilinear Pairings

Johan Wallén

February 4, 2003

Motivation

In standard ElGamal-style signature schemes such as DSA, the signature consist of two group elements.

Several applications need short signatures, e.g. in low-bandwidth environments or when the user have to type in the signature by hand.

Can we construct a discrete log or Diffie-Hellman based signature scheme where the signature is only one group element?

First solution in [Boneh, Lynn, and Shacham, 2001]. Their scheme works in any **Gap Diffie-Hellman group**—that is, a group where decisional DH is easy but computational DH is hard.

One of the first constructive applications of bilinear pairings.

Gap Diffie-Hellman groups

Let G be a cyclic (multiplicative) group of prime order p.

The **computational Diffie-Hellman (CDH)** problem is given (g, g^a, g^b) to compute g^{ab}.

The **decisional Diffie-Hellman (DDH)** problem is to distinguish between the distributions (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) for uniformly distributed $a, b, c \in \mathbb{Z}_p^*$.

G is a **Gap Diffie-Hellman (GDH)** group if DDH is easy but CDH is hard in G.

All popular GDH candidates arise from bilinear pairings.

Gap Diffie-Hellman groups (cont)

We assume that time is normalised to that one group operation takes unit time.

G is a τ**-decision group** for Diffie-Hellman if DDH can be solved in time τ.

The **advantage** of an algorithm A in solving CDH in G is

$$\mathrm{adv}_a^G = \Pr[a, b \leftarrow \mathbb{Z}_p^* : A(g, g^a, g^b) = g^{ab}] \ .$$

A is said to (t, ϵ)**-break** CDH if A runs in time t and $\mathrm{adv}_A^G \geq \epsilon$.

G is a (τ, t, ϵ)**-GDH group** if it is a τ-decision group and no algorithm (t, ϵ)-break CDH in G.

The gap signature scheme

Fix a GDH group G and a generator g.

Let $h\colon \{0,1\}^* \to G^*$ be a full domain hash function. The security analysis views h as a uniformly selected function $\{0,1\}^* \to G^*$.

Key generation. Pick a random $x \in \mathbb{Z}_p^*$ and compute $v = g^x$. The secret key is x and the public key is v.

Signing. Given a secret key x and message M, compute $m = h(M)$ and $\sigma = m^x$. The signature is $\sigma \in G^*$.

Verification. Given a pubic key v and a message M, compute $m = h(M)$ and verify that (g, v, m, σ) is a valid Diffie-Hellman tuple.

The gap signature scheme (cont)

If the signature is correct computed, $m = h(M) = g^a$ for some $a \in G^*$ and $(g, v, m, \sigma) = (g, g^x, g^a, g^{ax})$ indeed is a valid Diffie-Hellman tuple.

Key generation and signing is very fast and simple.

Verification is more complex, since it involves soling one DDH distance. For (τ, t, ϵ)-GDH groups, this takes τ steps.

The random oracle model

In the random oracle model [Bellare and Rogaway, 1993], we assume that both legitimate users and the adversary has oracle access to a random function.

The random oracle is then replaced with a cryptographic hash function.

Problem: the hash function is not random at all since it is a fixed function.

The random oracle model is in general unsound [Canetti, Goldreich, and Halevi, 1998]: there are protocols that are secure in the random oracle model but that has no secure implementation whatsoever in the standard model.

Security definition

We will show that the gap signature scheme is secure against **existential forgery** under **chosen-message attacks** in the **random oracle model**.

The forger F is given a public key v and oracle access to the signature scheme S and the hash function h. The forger can get signatures on any message of its choice.

Its goal is to produce a forged signature—that is, a message different from the signature queries and a valid signature to this message.

The **advantage** of F is

$$\mathrm{adv}_F^S = \Pr[(s, v) \leftarrow \mathrm{keyGen} : \mathrm{verify}(v, F^{S,h}(v))] \ .$$

If $(M, \sigma) = F^{S,h}(v)$, it is required that M is different from all S-queries.

Security definition (cont)

An algorithm F (t, q_h, q_s, ϵ)**-breaks** the signature scheme S, if F runs in time at most t, makes at most q_h hash queries, at most q_s signature queries, and $\mathrm{adv}_F^S \geq \epsilon$.

A signature scheme is (t, g_h, g_s, ϵ)**-secure** against existential forgery under chosen message attacks in the random oracle model if no algorithm (t, q_h, q_s, ϵ)-breaks it.

Security theorem

Let G be a (τ', t', ϵ')-GDH group. Then the gap signature scheme in G is (t, q_h, q_s, ϵ)-secure against existential forgery under a chosen plaintext attack in the random oracle model, where

$$t \leq t' - 2c \log|G|(q_h + q_s) \ ,$$
$$\epsilon \leq 2eq_s\epsilon' \ ,$$

and c is a small constant.

Security proof: overview

Assume that F (t, q_h, q_s, ϵ)-breaks the gap signature scheme. We will use F to construct an algorithm A that (t', ϵ')-breaks CDH in G.

A is given a CDH instance (g, g^a, g^b) and is supposed to compute g^{ab}. A uses the instance to create a public key v and emulates and attack of F on v.

The oracle calls of F are answered by A in a faked way.

When F outputs a forgery, A uses it to compute a solution to the CDH instance.

Security proof: overview (cont)

For convenience, we assume that F is well-behaved in the following sense.

We assume that F always requests a hash of M before it requests a signature of M and that it always asks for the hash of the message it outputs as its forgery.

Any forger F can easily be modified to have this property.

(But does this modification affect the bounds on t and ϵ?)

Security proof: overview (cont)

A maintains a list of the oracle queries of F and corresponding answers.

When a message M first arrives from F, it is assigned an index i.

Note that i is bounded by q_h.

The message is stored in M_i, its hash in m_i, and its signature (if available) in σ_i.

Constructing A

Instead of presenting the algorithm A directly, we will describe six increasingly sophisticated variants of A. The last one A_6 is the A we are looking for.

The first few variants will be completely trivial. They will just lay out a framework and simplify the analysis.

All variants except A_6 will only output $\top$ or $\perp$ to indicate success. When A_6 succeeds, it will output a solution to the CDH instance.

The algorithms are parametrised by a constant $0 \leq \zeta \leq 1$. Let B_ζ be the probability distribution that is 1 with probability ζ and 0 with probability $1 - \zeta$.

Variant 1

A_6 just emulates F.

Given (g, g^a, g^b), it sets the public key to $v = g^a$.

For $1 \leq i \leq q_h$, it picks a random bit $s_i \leftarrow B_\zeta$ and a random number $r_i \leftarrow \mathbb{Z}_p^*$. It sets $m_i = g^{r_i}$ and $\sigma_i = (g^a)^{r_i}$.

The values s_i are not used by A_1, but are important for the later variants.

Note that (g, g^a, m_i, σ_i) is a valid Diffie-Hellman tuple and thus σ_i is a valid signature for every message whose hash is m_i.

A runs F on input v. A answers hash queries for M_i with m_i and signature queries for M_i with σ_i. When F halts, A_1 outputs $\top$ if F succeeded in forging and outputs $\perp$ otherwise.

Variant 1 (cont)

Since the hashes m_i are uniformly distributed in G^*, A_1 answers hash queries as a random oracle.

Since σ_i is a valid signature for m_i, A_1 answers signature queries exactly as the real signature oracle would.

Thus, F behaves exactly as it would in a normal execution and

$$\Pr[A_1 = \top] = \mathrm{adv}_F^S \geq \epsilon \ .$$

Variant 2

A_2 introduces a small reduction in the success probability. This allows us to deviate from perfect simulation.

A_2 operates as A_1, but with one exception.

If F succeeds and outputs a forgery (M, σ), A_2 outputs $\top$ if $s_i = 1$, where i is the index of M. In all other cases A_2 output $\perp$.

Clearly, this does not affect the behaviour of F, so

$$\mathrm{Pr}[A_2 = \top] = \mathrm{adv}_F^S \cdot \mathrm{Pr}[s_i = 1] \geq \zeta\epsilon \ .$$

Variant 3

A_3 introduces another small reduction in the success probability.

A_3 operates as A_2 with the following exception. If F succeeds in finding a forgery (M, σ), A_3 outputs $\top$ if $M = M_i$, $s_i = 1$ and F asked for signatures only of messages M_j with $s_j = 0$.

Again, this does not affect the behaviour of F. If F made k signature queries,

$$\Pr[A_3 = \top] = \mathrm{adv}_F^S \cdot \zeta(1-\zeta)^k \geq (1-\zeta)^{q_s}\zeta\epsilon \ .$$

The final version A_6 will have the same success probability as A_3.

Variant 4

A_4 introduces a simple shortcut.

A_4 operates as A_3, but if F asks for a signature of a message M_j with $s_j = 1$, A_4 immediately halts with output $\perp$.

This shortcut does not affect the output distribution of A, so

$$\Pr[A_4 = \top] \geq (1 - \zeta)^{q_s} \zeta \epsilon \ .$$

Variant 5

A_5 changes the setup to better match the Diffie-Hellman instance (g, g^a, g^b). Otherwise, it is exactly as A_4.

In the setup stage, A_5 sets $m_i = g^b g^{r_i}$ and $\sigma_i = \perp$ when $s_i = 1$, and sets $m_i = g^{r_i}$ and $\sigma_i = (g^a)^{r_i}$ (as A_4) when $s_i = 0$.

Since g^{r_i} and $g^b g^{r_i}$ are both uniformly distributed in G^*, and A_4 never actually answers a signature query for a message M_j with $s_j = 1$, this does not affect the output distribution of A. Thus,

$$\Pr[A_5 = \top] \geq (1 - \zeta)^{q_s} \zeta \epsilon \ .$$

Variant 6

A_6 finally solves the Diffie-Hellman instance.

A_6 operates as A_5, but in those cases where A_5 outputs $\top$, A_6 outputs $\sigma(g^a)^{-r_i}$, where (M, σ) is the forgery and $M = M_i$. Clearly,

$$\Pr[A_6 \neq \perp] = \Pr[A_5 = \top] \geq (1-\zeta)^{q_s}\zeta\epsilon \ .$$

Moreover, A_6 succeeds only if $s_i = 1$, which means that $m_i = g^b g^{r_i} = g^{b+r_i}$.

Since σ is a valid signature of M, (g, g^a, m_i, σ) is a valid DH tuple, so $\sigma = g^{a(b+r_i)} = g^{ab}g^{ar_i}$.

Thus, the output of A_6 is g^{ab} when it succeeds. That is,

$$\mathrm{adv}^G_{A_6} = \Pr[A_6(g, g^a, g^b) = g^{ab}] \geq (1-\zeta)^{q_s}\zeta\epsilon \ .$$

Putting everything together

The algorithm $A = A_6$ solves the CDH problem with probability at least $(1-\zeta)^{q_s}\zeta\epsilon$ for arbitrary $\zeta \in [0,1]$. The bound is maximised for $\zeta = \frac{1}{q_s+1}$, where it gives the bound $\epsilon' \geq \frac{\epsilon}{q_s}\left(1 - \frac{1}{q_s+1}\right)^{q_s+1}$. Since $\left(1 - \frac{1}{q_s+1}\right)^{q_s+1} \geq \frac{1}{2e}$ for $q_s > 0$, this gives

$$\epsilon \leq q_s\epsilon' \cdot 2e \ .$$

Putting everything together (cont)

The running time of A includes the running time t of F. In addition, it has to do one exponentiation for each oracle query. Each exponentiation can be done using $2 \log p$ group operations. Thus, the running time of A is

$$t' \leq t + 2c \log p(q_s + q_h) \ ,$$

where c is a small constant accounting for the remaining overhead of A.

This completes the security proof. $\square$

Conclusions

The gap signature scheme is a very nice constructive example of how bilinear pairings can be used in cryptography.

The resulting gap signature scheme is simple with a very efficient signature algorithm and shot signatures (160 bits for a security comparable with 320-bit DSA or 1024-bit RSA signatures).

The security proof is also fairly intuitive and simple.

Note that the signature scheme is completely deterministic.

References

Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First Annual Conference on Computer and Communications Security*. ACM, 1993.

Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology—Asiacrypt 2001*, volume 2248 of *LNCS*, pages 514–532. Springer-Verlag, 2001.

Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *30th ACM Symposium on the Theory of Computation*, pages 209–218. ACM, 1998. A revisited version is available from `http://www.wisdom.weizmann.ac.il/~oded/`.