[PS 96] – Security Proofs for Signature Schemes. Pointcheval and Stern. Eurocrypt 1996.

[BN06] – Multi-Signatures in Plain Public Key Model and a General Forking Lemma. Ballare and Neven. ACM 2006

[BCCGP16] – Efficient zero-knowledge Arguments for arithmetic circuits in the Discrete Logarithm setting. Bootle, Cerulle, Chaidos, Groth and Petit. Eurocrypt 2016.

[BBBPWM18] – "Bulletproofs: Short proofs for confidential transactions and More", Bünz, Bootle, Boneh, Poelstra, Wuille and Maxwell, IEEE 2018

Why should I care about this topic ?

usually/sometimes

→ We wish to prove security by stating something like this:

"If ∃ Adversary that breaks my protocol. Then I can break this believed-to-be hard math problem"

Ex dlog, factoring, DDH etc...

Not always straight forward

↑

Forking lemma – a tool to achieve above

Intuitively: if A can break my protocol once he/she can do it twice. And I can break the believed-to-be-hard math problem if I recieve 2 forgeries of the protocol.

↑

with same input but different response from the Random Oracle

↑

because if only possible with one particular response from the R.O then it is so unlikely to get this precise response that I say it is negligiable

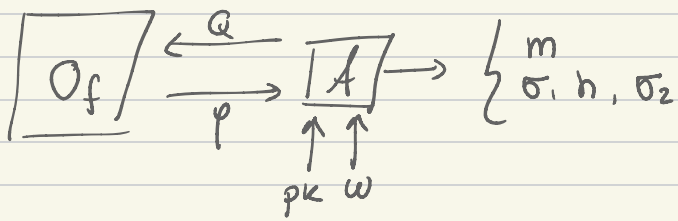
Forking lemma

Initially introduced by [PS96]

$Sign(m, sk) \rightarrow (\sigma_1, h, \sigma_2)$

- $h = \mathcal{H}(\sigma_1, m)$
- σ_2 depends only on σ_1, m, h
- Sometimes σ_1 & h can be omitted

No message attack



Lemma "The forking lemma" [PS96] [PS96]

Let A be a probabilistic PPT Turing machine given only the public data as input. If A can find, with non-negligible probability a valid signature $(m, \sigma_1, h, \sigma_2)$, then with non-negligible probability a replay of this machine with the same random tape and a different oracle outputs two valid signatures $(m, \sigma_1, h, \sigma_2)$ & $(m, \sigma_1, h', \sigma_2')$ such that $h \neq h'$.

If only two → one must be at least $\varepsilon/2$ likely.

proof of lemma

Helping lemma: Let $A \subset X \times Y$, such that $\Pr[A(x,y)] \geq \varepsilon$, then there exists $\Omega \subset X$ s.t:

(i) $\Pr[x \in \Omega] \geq \varepsilon/2$

(ii) whenever $a \in \Omega$, $\Pr[A(a,y)] \geq \varepsilon/2$

Assume **no msg attacker**

A - PPT TM with random tape ω.

During the attack this machine $q = poly(\lambda)$ questions to a RO O_f

Let $\begin{cases} Q_1 \dots Q_q \text{ be the questions} \\ \rho_1 \dots \rho_q \text{ be the answers} \end{cases}$

Random choice of the oracle O_f => random choice of $\rho_1 \dots \rho_q$

assumption

For a random choice of $\omega\, \rho_1 \dots \rho_q$, with non-negl prob A outputs a valid signature $(m, \sigma_1, h, \sigma_2)$

Remarks
- It is easy to see that $\Pr[(m,\sigma_1) \notin \{Q_i\}_{i=1}^q] \leq$ negl due to $h = O_f(m, \sigma_1)$.

$\Pr[\text{success}] \geq$ negl

$\Pr[Q_\beta] \geq$ negl

For $\beta \in \{1 \dots q\}$ $\exists$ polynomial P s.t probability of success over $\omega, \rho_1 \dots \rho_q$ with $Q_\beta = (m, \sigma_1)$ is $\geq 1/P(\lambda)$

help lemma

Given such a β: $\xRightarrow{\text{lemma}}$ $\exists$ a non-negl subset of Ω_β of good ω's for such a good ω the probability of success over $\rho_1, \dots, \rho_q$ with $Q_\beta = (m, \sigma_1) \geq 1/2P(\lambda)$.

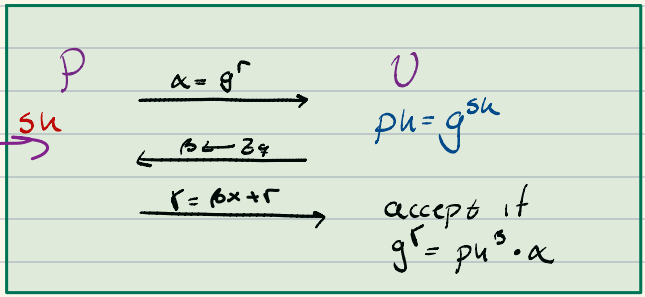
applying lemma again

Given such β and ω $\xRightarrow{\text{lemma}}$ $\exists$ a non-negl subset $R_{\beta,\omega}$ of 'good' $(\rho_1, \dots, \rho_{\beta-1})$'s. For such "good" answers the probability of success of the attacker over $\rho_\beta, \dots, \rho_q$ with $Q_\beta = (m, \sigma_1)$ is $\geq \frac{1}{2} \cdot \frac{1}{2P(\lambda)} = \frac{1}{4P(\lambda)}$

Then with such β, ω & $(\rho_1 \dots \rho_{\beta-1})$ if we randomly choose $\rho_\beta, \dots \rho_q$ & $(\rho'_\beta, \dots, \rho'_q)$, with a non negligable probability we obtain two valid signatures $\begin{cases} (m, \sigma_1, h, \sigma_2) \\ (m, \sigma_1, h', \sigma'_2) \end{cases}$ s.t $h \neq h'$ ($k(\lambda) \gg \log \lambda$)

Finally with random choice of $\begin{cases} \beta, \omega, \rho_1, \dots, \rho_{\beta-1}, \rho_\beta, \dots \rho_q \\ \beta, \omega \quad \rho_1, \dots, \rho_{\beta-1}, \rho'_\beta, \dots, \rho'_q \end{cases}$ with non-negl prob 2 valid signatures.

Special Soundness + Schnorr ID Scheme



Rewinding P to here!

Remark α the same in both transcripts.

Given 2 accepted transcript $\begin{cases} tr = (\alpha, \beta, \gamma) \\ tr' = (\alpha, \beta', \gamma') \end{cases}$

we can extract x:

if $\beta' \neq \beta$ then $x = \dfrac{\gamma - \gamma'}{\beta - \beta'}$.

2 generalizations of the FL

[BN06] - generalizes the lemma to be more generic than a proof technique for Digital signatures.

Rewinds once

[BCCGP16] - generalizes to consider more than one rewinding. Instead they consider an execution tree.

Rewinds as many times as needed

- unbounded runtime of extracter but expected runtime is $T' \approx T$

T: Runtime of prover.

[BN06]

Generalized Forking Lemma

moving away from signatures

moving away from signatures.
This more general notion doesn't mention { - RO, - signatures
instead it focus on the output behaviour of an algorithm when run twice on related inputs.
This makes it applicable to other contexts than signature schemes & seperates the probabilistic analysis of the rewinding from the actual simulation in the security proof

Notation x - public key
$(h_1, \dots, h_q)$ - replies to queries to a R.O.

queries to RO — Hash function image space — by ρ

LEMMA 1. **[General Forking Lemma]** *Fix an integer $q \geq 1$ and a set H of size $h \geq 2$. Let A be a randomized algorithm that on input $x, h_1, \dots, h_q$ returns a pair, the first element of which is an integer in the range $0, \dots, q$ and the second element of which we refer to as a side output. Let IG be a randomized algorithm that we call the input generator. The accepting probability of A, denoted* acc, *is defined as the probability that $J \geq 1$ in the experiment*

"public key"

$$x \xleftarrow{\$} \mathsf{IG}\,;\ h_1, \dots, h_q \xleftarrow{\$} H\,;\ (J, \sigma) \xleftarrow{\$} \mathsf{A}(x, h_1, \dots, h_q)\,.$$

The forking algorithm $\mathsf{F_A}$ *associated to A is the randomized algorithm that takes input x proceeds as follows:*

Forking algorithm, only takes public key x as input.

Algorithm $\mathsf{F_A}(x)$
Pick coins ρ for A at random
$h_1, \dots, h_q \xleftarrow{\$} H$
$(I, \sigma) \leftarrow \mathsf{A}(x, h_1, \dots, h_q; \rho)$ — what is ε?
If $I = 0$ then return $(0, \varepsilon, \varepsilon)$
Forking point I.
same until here
$h'_I, \dots, h'_q \xleftarrow{\$} H$
$(I', \sigma') \leftarrow \mathsf{A}(x, h_1, \dots, h_{I-1}, h'_I, \dots, h'_q; \rho)$ — same, different
If $(I = I'$ and $h_I \neq h'_I)$ then return $(1, \sigma, \sigma')$
Else return $(0, \varepsilon, \varepsilon)$. — making sure a forking happened

Let

$$\mathrm{frk} \;=\; \Pr\left[\, b = 1 \;:\; x \xleftarrow{\$} \mathsf{IG}\,;\ (b, \sigma, \sigma') \xleftarrow{\$} \mathsf{F_A}(x) \,\right].$$

Then

$$\mathrm{frk} \;\geq\; \mathrm{acc} \cdot \left(\frac{\mathrm{acc}}{q} - \frac{1}{h} \right). \qquad (1)$$

queries to R.O. — size of $|H|$.

Alternatively,

$$\mathrm{acc} \;\leq\; \frac{q}{h} + \sqrt{q \cdot \mathrm{frk}}\,. \quad \blacksquare \qquad (2)$$

The transform can be used only if the protocol is **public coin**, that is when the random generated by the verifier is also known by the prover in the interactive version of the protocol.

[BCCG16] # Generalized Forking Lemma

Tree of executions

The setting

Randomness gen by verifier is unknown by prover

- $(2\mu+1)$-move public coin
- μ challanges $x_1, \dots x_\mu$ in sequence
- $n_i \geq 1 \quad \forall i \in \{1 \dots \mu\}$
- $\prod n_i$ accepting transcripts with challanges in the following tree:
  - depth μ
  - $\prod n_i$ leaves
  - root is labelled with the statement
  - Each node at depth i has n_i children each labelled with a distinct value for the i^{th} challange

Definition 7 (Statistical witness-extended emulation). $(\mathcal{P}, \mathcal{V})$ *has* statistical witness-extended emulation *if for all deterministic polynomial time* $\mathcal{P}^*$ *there exists an expected polynomial time emulator* $\mathcal{E}$ *such that for all interactive adversaries* $\mathcal{A}$

$$\Pr\left[(u,s) \leftarrow \mathcal{A}(1^\lambda); tr \leftarrow \langle \mathcal{P}^*(u,s), \mathcal{V}(u)\rangle : \mathcal{A}(tr) = 1\right]$$
$$\approx \Pr\left[\begin{array}{l}(u,s) \leftarrow \mathcal{A}(1^\lambda); (tr,w) \leftarrow \mathcal{E}^{\langle \mathcal{P}^*(u,s), \mathcal{V}(u)\rangle}(u) : \\ \mathcal{A}(tr) = 1 \text{ and if } tr \text{ is accepting then } (u,w) \in R\end{array}\right]$$

where the oracle called by $\mathcal{E}^{\langle \mathcal{P}^*(u,s), \mathcal{V}(u)\rangle}$ *permits rewinding to a specific point and resuming with fresh randomness for the verifier from this point onwards.*

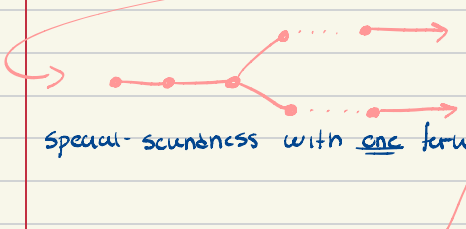
Given an Adversary that produces an accepting argument with some probability, $\exists$ an emulator that produces a similar argument with some probability together with a witness w.

The emulator may rewind Prover/Verifier to any previous move.

Definition 8 (Public coin). *An argument* $(\mathcal{P}, \mathcal{V})$ *is called* public coin *if the verifier chooses his messages uniformly at random and independently of the messages sent by the prover, i.e., the challenges correspond to the verifier's randomness* ρ.

Above tree we referred to as an $(n_1, \dots, n_\mu)$-tree of accepting transcripts.

generalisation of Special-Soundness for sigma protocols where $\mu = 1$, $n = 2$



special-soundness with one term

the $n_1 \dots n_\mu$-tree. $\&$ generalization

The Forking Lemma [BCCGP16]

Let (P, V) be a $(2\mu+1)$-move public coin interactive protocol. Let X be a witness extraction algorithm that always succeeds in extracting a witness from an $(n_1, \dots, n_\mu)$-tree of accepting transcripts in PPT. Assume $\prod_{i=1}^{\mu} n_i \in \text{poly}(\lambda)$ (security parameter)

Then (P, V) has (statistical) witness-extended emulation.

Application of Generalized Forking Lemma

Bulletproofs [BBBPWM18]

Inner product argument:

An efficient proof system of the following relation:

$$\{ g, h \in \mathbb{G}^n, P \in \mathbb{G}, c \in \mathbb{Z}_q : a, b \in \mathbb{Z}_q^n : P = g^a h^b \wedge c = \langle a, b \rangle \}$$

(input only known by Prover)

(g, h: common input to both prover & verifier)

The inner product protocol

(slight modification)

Input: $(\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, u, P \in \mathbb{G} ; \mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n)$ (7)
$\mathcal{P}_{\mathsf{IP}}$'s input: $(\mathbf{g}, \mathbf{h}, u, P, \mathbf{a}, \mathbf{b})$ (8)
$\mathcal{V}_{\mathsf{IP}}$'s input: $(\mathbf{g}, \mathbf{h}, u, P)$ (9)
Output: {$\mathcal{V}_{\mathsf{IP}}$ accepts or $\mathcal{V}_{\mathsf{IP}}$ rejects} (10)
if $n = 1$: (11)
$\mathcal{P}_{\mathsf{IP}} \rightarrow \mathcal{V}_{\mathsf{IP}} : a, b$ (12)
$c = a \cdot b$ (13)
$\mathcal{V}_{\mathsf{IP}}$ checks if $P = g^a h^b u^c$: (14)
if yes, $\mathcal{V}_{\mathsf{IP}}$ accepts (15)
otherwise, $\mathcal{V}_{\mathsf{IP}}$ rejects (16)
else: $(n > 1)$ (17)
$\mathcal{P}_{\mathsf{IP}}$ computes: (18)
$n' = \frac{n}{2}$ (19)
$c_L = \langle \mathbf{a}_{[:n']}, \mathbf{b}_{[n':]} \rangle \in \mathbb{Z}_p$ (20)
$c_R = \langle \mathbf{a}_{[n':]}, \mathbf{b}_{[:n']} \rangle \in \mathbb{Z}_p$ (21)
$L = \mathbf{g}_{[n':]}^{\mathbf{a}_{[:n']}} \mathbf{h}_{[:n']}^{\mathbf{b}_{[n':]}} u^{c_L} \in \mathbb{G}$ (22)
$R = \mathbf{g}_{[:n']}^{\mathbf{a}_{[n':]}} \mathbf{h}_{[n':]}^{\mathbf{b}_{[:n']}} u^{c_R} \in \mathbb{G}$ (23)
$\mathcal{P}_{\mathsf{IP}} \rightarrow \mathcal{V}_{\mathsf{IP}} : L, R$ (24)
$\mathcal{V}_{\mathsf{IP}} : x \xleftarrow{\$} \mathbb{Z}_p^*$ (25)
$\mathcal{V}_{\mathsf{IP}} \rightarrow \mathcal{P}_{\mathsf{IP}} : x$ (26)
$\mathcal{P}_{\mathsf{IP}}$ and $\mathcal{V}_{\mathsf{IP}}$ compute: (27)
$\mathbf{g}' = \mathbf{g}_{[:n']}^{x^{-1}} \circ \mathbf{g}_{[n':]}^{x} \in \mathbb{G}^{n'}$ (28)
$\mathbf{h}' = \mathbf{h}_{[:n']}^{x} \circ \mathbf{h}_{[n':]}^{x^{-1}} \in \mathbb{G}^{n'}$ (29)
$P' = L^{x^2} P R^{x^{-2}} \in \mathbb{G}$ (30)
$\mathcal{P}_{\mathsf{IP}}$ computes: (31)
$\mathbf{a}' = \mathbf{a}_{[:n']} \cdot x + \mathbf{a}_{[n':]} \cdot x^{-1} \in \mathbb{Z}_p^{n'}$ (32)
$\mathbf{b}' = \mathbf{b}_{[:n']} \cdot x^{-1} + \mathbf{b}_{[n':]} \cdot x \in \mathbb{Z}_p^{n'}$ (33)
recursively run Protocol 2 on input (34)
$(\mathbf{g}', \mathbf{h}', u, P'; \mathbf{a}', \mathbf{b}')$ (35)

(Rewinds this part 3 times)

Protocol 2: Improved Inner-Product Argument

Sketch of the security proof

only the parts relevant for forking lemma

An recursive argument that in each round we either extract a witness or a discrete logarithm relation.

$n = |g| = 1$: prover reveals the witness.

$n > 1$: Extracter runs prover to get L, R, then using 3 different challanges (x_1, x_2, x_3), the extracter obtains: (pink part in protocol)

$a_1^{(i)}, b_1^{(i)}, a_2^{(i)}, b_2^{(i)}, a_3^{(i)}, b_3^{(i)}$ s.t

$$L^{x_i^2} P R^{x_i^{-2}} = g_1^{a_1^{(i)}} h^{b_1^{(i)}} u^{\langle a_1^{(i)}, b_1^{(i)} \rangle} \quad \forall i = 1, 2, 3$$

From these 3 equalities the extracter can extract either a witness or a non-trivial discrete logarithm between g_1, h & u.