

Forschungsinstitut
Cyber Defence
Universität der Bundeswehr München



Asynchronous Remote Key Generation (ARKG) and Its Applications

Mark Manulis

PACY Lab, Research Institute CODE

Universität der Bundeswehr München

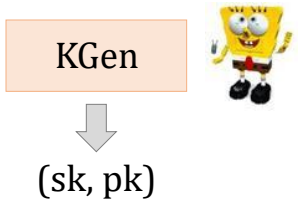
Based on joint works with Nick Frymann, Daniel Gardham, Franziskus Kiefer, Emil Lundberg, Hugo Nartz, Dain Nilson

Key Encapsulation Mechanism

PACY LAB

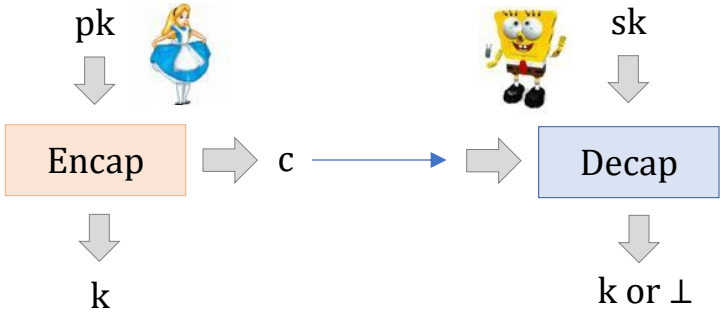
KEMs

- Standard building block in cryptography
- Sender encapsulates symmetric key k for receiver
  - At the end sender and receiver share k
- Symmetric keys “look like” random keys
- k can be used in arbitrary symmetric protocols (AEAD, MAC)



Some applications of KEMs

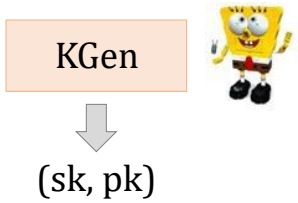
- Hybrid encryption, i.e., KEM/DEM framework
- Key establishment
- Secure channels and messaging, esp. post-quantum
  - TLS
  - Two party messaging
  - Group messaging, e.g. OpenMLS



ARKG vs KEM

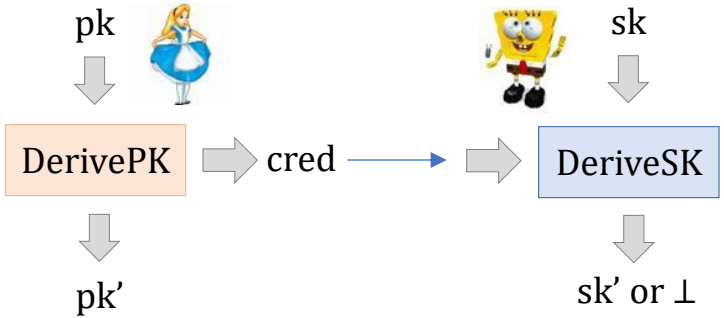
Asynchronous Remote Key Generation (ARKG)

- Asymmetric analogy of KEMs [Fry+20@CCS20]
- Sender derives new *public* keys pk' for receiver
- Receiver later derives corresponding *private* keys sk'
- Derived key pairs "look like" freshly generated
- (sk' pk') can be used in arbitrary public key protocols



ARKG algorithms

- Setup(1^λ) → pp
- KGen(pp) → (sk, pk)
- DerivePK(pk, aux) → (pk', cred)
- DeriveSK(sk, cred) → sk' / ⊥
- Check(sk', pk'): true / false



Research landscape around ARKG

PACY LAB

ARKG constructions

- [Fry+20@CCS20]: generation of ECDL key pairs, scheme based on ECDH
- [Fry+23@ACNS23]: generation of key pairs for PbC, uses ϕ-AKG abstraction and Uber assumption for different key structures
- [FryGM23@EuroS&P23]: generation of LWE-based key pairs, scheme based on Split KEMs
- [BreCF23@eprint/2023/1275]: generic with LWE instances based on anonymous KEMs
- [SteW24@ASIACCS24]: generic with LWE instances based on KEMs with blinding properties
- Ongoing IETF standardisation by Yubico: https://github.com/Yubico/arkg-rfc

Applications

- Backup/recovery in WebAuthn:
  - [Fry+20, BreCF23, SteW24]
- Delegation of WebAuthn accounts:
  - [FryGM22@ESORICS22] incl. new proxy signatures with unlinkable warrants
- ARKG techniques in blockchains
  - [Wuille12]: deterministic wallets
  - [Yu20]: stealth addresses
  - [Pu+23@CCS23]: stealth signatures
- Ongoing work in the context of European digital ID framework

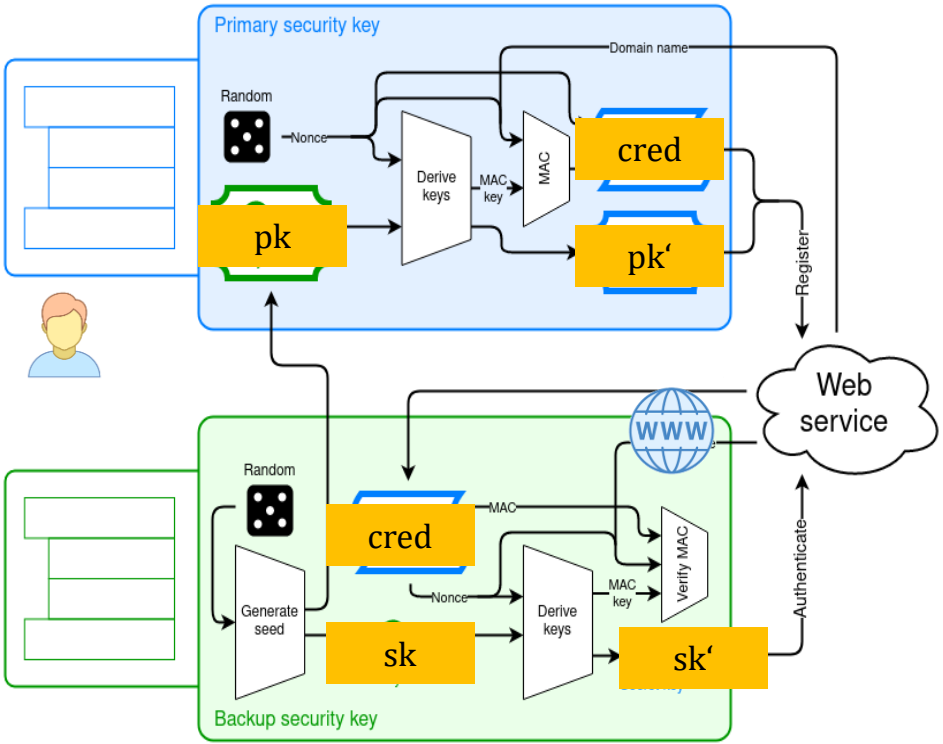
ARKG in WebAuth Recovery

yubico

PACY LAB

Main idea

- Two authenticators: primary (PA), backup (BA)
- Initial step: BA has a long-term public key pk that it sends to PA
- PA co-registers *derived* public keys pk' on behalf of BA on user's accounts
- BA recovers corresponding private keys sk' at a later time when authenticating



©Yubico's blog

ARKG security properties

PACY LAB

ARKG algorithms

- Setup(1^λ) → pp
- KGen(pp) → (sk, pk)
- DerivePK(pk, aux) → (pk', cred)
- DeriveSK(sk, cred) → sk' / ⊥
- Check(sk', pk'): true / false

SK-security (key secrecy) of ARKG

- $\mathcal{A}$ can't generate (sk', pk', cred)
- (honest, malicious) x (weak, strong)
- Implications: ms ⇒ mw ⇒ hw and ms ⇒ hs ⇒ hw

PK-unlinkability of ARKG

- $\mathcal{A}$ can't decide whether (sk', pk') was generated using pk_0 or pk_1
- [BreCF23, SteW24] : $\mathcal{A}$ also has access to cred

$\mathrm{Exp}^{\mathrm{ks}}_{\mathrm{ARKG},\mathcal{A}}(\lambda)$

1 : $\mathrm{pp} \leftarrow \mathrm{Setup}(1^\lambda)$
2 : $(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathrm{KGen}(\mathrm{pp})$
3 : $(\mathrm{sk}^\star, \mathrm{pk}^\star, \mathrm{cred}^\star) \leftarrow \mathcal{A}^{O_{\mathrm{pk}'}, O_{\mathrm{sk}'}}(\mathrm{pp}, \mathrm{pk})$
4 : $\mathrm{sk}' \leftarrow \mathrm{DeriveSK}(\mathrm{pp}, \mathrm{sk}, \mathrm{cred}^\star)$
5 : **return** $\mathrm{Check}(\mathrm{sk}^\star, \mathrm{pk}^\star) \stackrel{?}{=} 1$
6 : $\wedge\ \mathrm{Check}(\mathrm{sk}', \mathrm{pk}^\star) \stackrel{?}{=} 1$
7 : $\wedge\ \mathrm{cred}^\star \notin \mathrm{SKList}$
8 : $\wedge\ (\mathrm{pk}^\star, \mathrm{cred}^\star) \in \mathrm{PKList}$

$\mathrm{Exp}^{\mathrm{pku}}_{\mathrm{ARKG},\mathcal{A}}(\lambda)$

1 : $\mathrm{pp} \leftarrow \mathrm{Setup}(1^\lambda)$
2 : $(\mathrm{sk}_0, \mathrm{pk}_0) \leftarrow \mathrm{KGen}(\mathrm{pp})$
3 : $b \leftarrow\$ \{0, 1\}$
4 : $b' \leftarrow \mathcal{A}^{O^b_{\mathrm{pk}'}}(\mathrm{pp}, \mathrm{pk}_0)$
5 : **return** $b \stackrel{?}{=} b'$

ARKG for DL-based key pairs

ARKG scheme for DL-based Key Pairs

PACY LAB

Background on the DL-ARKG scheme

- The origin of the protocol can be traced back to the techniques used in stealth addresses.
- In 2019 Yubico wanted to use it for account recovery in WebAuthn and approached us for security analysis.

$pp = ((\mathbb{G}, g, q), MAC, KDF_1, KDF_2)$

$(sk, pk) = (s, S = g^s) \leftarrow KGen$

DerivePK

$(e, E = g^e) \leftarrow KGen$



$ck \leftarrow KDF_1(S^e)$

$mk \leftarrow KDF_2(S^e)$

$P = S \times g^{ck}$ $(= g^{s+ck})$

$\mu \leftarrow MAC.Tag(mk, (E, aux))$

return pk' = P and cred = (E, aux, μ)

cred →



DeriveSK

$ck \leftarrow KDF_1(E^s)$

$mk \leftarrow KDF_2(E^s)$

if MAC.Verify(mk, (E, aux), μ) = true

then return $sk' = s + ck \pmod q$

else return ⊥

Security of DL-ARKG keys

SK-security

- malicious-strong (and malicious-weak) based on snPRF-ODH assumption in random oracle model
- honest-strong (and honest-weak) based on DL assumption in the standard model

PK-unlinkability

- based on nnPRF-ODH and DL assumptions in random oracle model
- if $\mathcal{A}$ has access to cred then we also need pseudorandomness of the MAC

Remark on the use of **PRF-ODH** assumption:

$pp = ((\mathbb{G}, g, q), g^u, g^v, PRF))$
$y_0 \leftarrow PRF(g^{uv}, x)$
y_1 is random
$b \in_R \{0,1\}$

pp, y_b, x



$\mathcal{A}$

ODH oracle (single call in snPRF-ODH)
Input: $G \in \mathbb{G}$ and $(G, x') \neq (g^v, x)$
Output: $y \leftarrow PRF(G^u, x')$

In case of ARKG-DL:
$g^u : S = g^s$ is pk
$g^v : E = g^e$
$y : ck \leftarrow KDF_1(S^e)$

ARKG for PbC using ϕ-AKG

Generalising ARKG using ϕ-AKG

PACY LAB

Observations

- In DL-ARKG: DerivePK uses S^e whereas DeriveSK uses E^s.
- These operations can be generalised using some suitable mapping $\phi : \mathbb{G}_{sk} \times \mathbb{G}_{pk} \rightarrow \mathbb{G}$.
- In DL-ARKG: $sk' = s + ck$ (in $\mathbb{Z}_q$).
- Assume $+$ in $\mathbb{Z}_q$ is an instance of some algorithm $\text{SKCombine}(sk_1, sk_2) = sk$ for $sk_1, sk_2, sk \in \mathbb{G}_{sk}$.

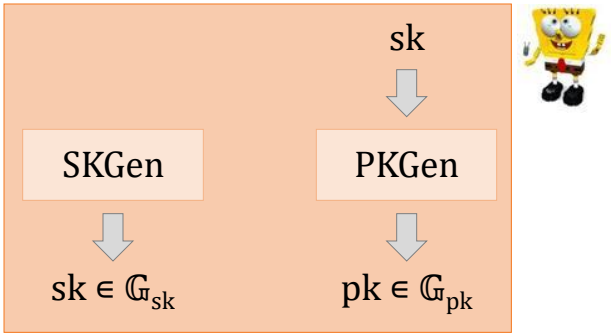
ϕ-AKG (Asymmetric Key Generation with mapping ϕ)

- $\text{Setup}(1^\lambda) \rightarrow pp$ describing $\mathbb{G}_{sk}$ and $\mathbb{G}_{pk}$
- $\text{SKGen}(pp) \rightarrow sk$ and $\text{PKGen}(sk) \rightarrow pk$
- Check(sk, pk): true / false

Efficient $\phi : \mathbb{G}_{sk} \times \mathbb{G}_{pk} \rightarrow \mathbb{G}$ such that $\phi(sk_1, pk_2) = \phi(sk_2, pk_1)$

- $\text{SKCombine}(sk_1, sk_2) = sk$
- $\text{SKInv}(sk_2, sk) = sk_1$

Traditional KGen is a combination:



Transforming ϕ-AKG into ARKG

PACY LAB

We can use DL-ARKG but replace all operations related to DL keys and groups with ϕ-AKG functionality.

pp = (AKG.Setup, MAC, KDF_1, KDF_2) where $KDF_1 : \{0,1\}^* \rightarrow \mathbb{G}_{sk}$

(sk, pk) = (s, S) ← AKG.KGen

DerivePK

(e, E) ← AKG.KGen

ck ← $KDF_1(\phi(S, e))$

mk ← $KDF_2(\phi(S, e))$

P = AKG.PKGen(ck) · S (group law in $\mathbb{G}_{pk}$)

μ ← MAC.Tag(mk, (E, aux))

return pk' = P and cred = (E, aux, μ)



cred →



DeriveSK

ck ← $KDF_1(\phi(s, E))$

mk ← $KDF_2(\phi(s, E))$

if MAC.Verify(mk, (E, aux), μ) = true

then return sk' = AKG.SKCombine(ck, s)

else return ⊥

PK-unlinkability and malicious-strong SK-security of generalised ARKG can be proven based on the security assumptions on ϕ-AKG and uses a generalised version of PRF-ODH assumption which we call $PRF\text{-}O_\phi$.

Pairing-based ϕ-AKG schemes

PACY LAB

- We focus on pairing-based ϕ-AKG schemes to obtain first pairing-based ARKG constructions.
- We use decisional Uber assumption to obtain ARKG for different key structures and pairing types.
- Bilinear maps $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Type 1 ($\mathbb{G}_1 = \mathbb{G}_2$). Type 2 (efficient $\gamma : \mathbb{G}_2 \to \mathbb{G}_1$). Type 3 ($\gamma$ is intractable).

Decisional Uber Assumption (Boyen 2008, implied by standard assumptions such as DBDH, XDH, SXDH)

- Given $(g_1^{\mathbf{F}(\mathbf{z})}, g_2^{\mathbf{H}(\mathbf{z})}, g_T^{\mathbf{K}(\mathbf{z})}, Z)$ decide whether $Z = g_T^{Q(\mathbf{z})}$ or Z is a random element in $\mathbb{G}_T$ where
  - $\mathbf{F} = (F_1, F_2, \ldots)$, $\mathbf{H} = (H_1, H_2, \ldots)$, $\mathbf{K} = (K_1, K_2, \ldots)$ are *vectors* of m-variate polynomials
  - Q is a target m-variate polynomial
  - $\mathbf{z}$ is a *vector* of random order elements of size m

- In pairing-based ϕ-AKG key pairs are parameterised by vectors $(\mathrm{sk}(\mathbf{x}), \mathrm{pk}(\mathbf{x}))$, $(\mathrm{sk}(\mathbf{y}), \mathrm{pk}(\mathbf{y}))$ and

$$\phi(\mathrm{sk}(\mathbf{x}), \mathrm{pk}(\mathbf{y})) = g_T^{Q(\mathbf{x}, \mathbf{y})}$$

- In the language of the Uber assumption $\mathrm{pk}(\mathbf{x})$, $\mathrm{pk}(\mathbf{y})$ are group elements and secret $\mathbf{z} = (\mathbf{x}, \mathbf{y})$.

Selected ϕ-AKG for Type-2/3 Pairings

PACY LAB

$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$

Key pair $(sk, pk) = (x, g_2^{\,x})$

- E.g. in **BLS-3** signatures.
- $\phi(sk(\mathbf{x}), pk(\mathbf{y})) = \phi(x, g_2^{\,y}) = e(g_1^{\,x}, g_2^{\,y})$ This assumes $Q(\mathbf{x}, \mathbf{y}) = xy$. SXDH assumption.
- Group law in $\mathbb{G}_{pk}$: $pk(\mathbf{x}) \cdot pk(\mathbf{y}) = g_2^{\,x} g_2^{\,y}$
- AKG.SKCombine$(sk(\mathbf{x}), sk(\mathbf{y})) = x + y$

Key pair $(sk, pk) = ((x_1, x_2), (g_1^{\,x1}, g_1^{\,x2}))$.

- E.g. in **Pointcheval-Sanders** signatures.
- $\phi((x_1, x_2), (g_1^{\,y1}, g_1^{\,y2})) = e(g_1^{\,y1}, g_2^{\,x1}) e(g_1^{\,y2}, g_2^{\,x2})$ This assumes $Q(\mathbf{x}, \mathbf{y}) = x_1y_1 + x_2y_2$. SXDH assumption.
- Group law in $\mathbb{G}_{pk}$: $pk(\mathbf{x}) \cdot pk(\mathbf{y}) = (g_1^{\,x1} g_1^{\,y1}, g_1^{\,x2} g_1^{\,y2})$
- AKG.SKCombine$(sk(\mathbf{x}), sk(\mathbf{y})) = (x_1 + y_1, x_2 + y_2)$

Further examples in the paper for key pairs compatible with CL, Waters signatures and structure-preserving signatures.

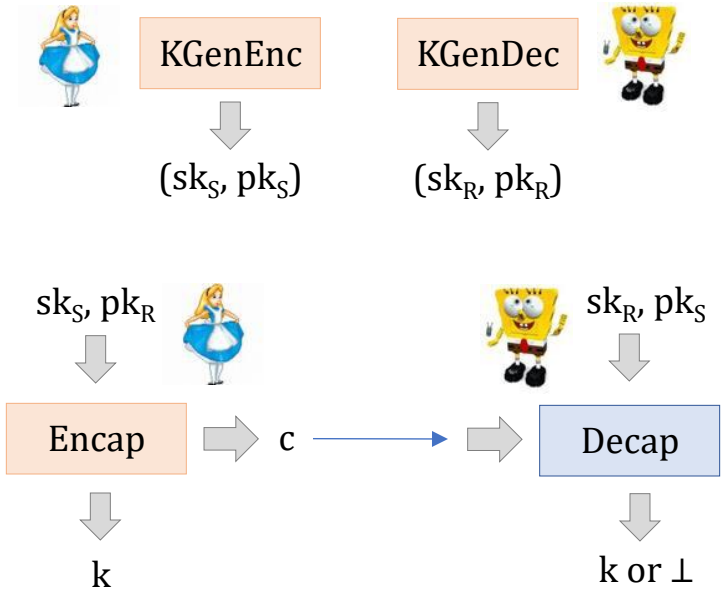
ARKG for (R)LWE key pairs using Split KEMs

ARKG for Lattice-based Cryptosystems

- For post-quantum ARKG we focus on lattice based constructions.
- We build ARKG generically from lattice-based Split KEMs.

Split KEMs [Bre+20]

- Like a KEM but both sender and receiver have key pairs.
- Sender encapsulates symmetric key k for the receiver.
- Receiver obtains k using own secret key and sender‘s public key.
- Can be used to „translate“ Diffie-Hellman protocols into KEM-based flow.



ARKG from LWE-based Split-KEMs

PACY LAB

- We build transformation using approach similar to DL-ARKG but with IND-CCA lattice-based Split KEMs.
- Key pairs in our lattice-based ARKG take form of (R)LWE instances: (sk, pk) = (S, AS + E)

pp = (sKEM.Setup, KDF)

(sk, pk) = (S, B) ← sKEM.KGenDec

DerivePK

(S', B') ← sKEM.KGenEnc



(K, c) ← sKEM.Encap(B, S')

μ ← KDF(K)

E' ← Gaussian(σ)

P = B + AK + E' (= A(S+K)+E+E')

return pk' = P and cred = (c, B', μ)

cred →



K ← sKEM.Decap(S, B', c)

μ' ← KDF(K)

if μ = μ' **then** S'' = S + K

if (S'' not rejected) **then return** sk' = S''

else return ⊥

DeriveSK

Rejection would lead to repetition of DerivePK.

Rejection sampling and τ-Correctness

Rejection sampling is needed to ensure that sk' output by DeriveSK is secure.

- sk' is accepted on a probability dependent on the desired distribution (i.e. a small centred gaussian).
- Sometimes even correct keys with small norm will be rejected.

τ-Correctness of ARKG

- In DL and pairing-based ARKG the correctness property is perfect.
- In lattice-based ARKG we need to relax it to account for possible failures in DeriveSK.
- τ-correctness: Prob[Check(sk', pk')] $> 1 - 2^{\tau}$

Note that lattice-based ARKG constructions are highly parameter dependent.

- Trade-off between τ and the *repetition rate* M, i.e. number of derived pk' needed to achieve τ-correctness.
- E.g., using Kyber (Level 5 parameters) we can get $\tau = 174$ for $M = 9$ executions of DerivePK.

Distributed ARKG

Appears at ACNS 2025

dARKG: Distributed ARKG

Motivation



Problem with ARKG sk‘ derived by a single proxy.

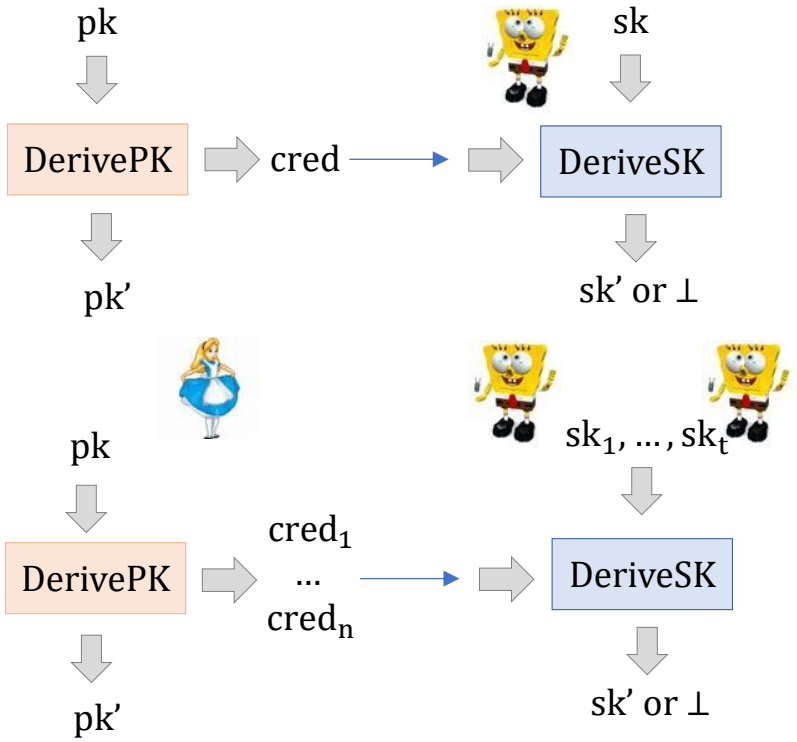
- **Single-point-of-failure**: risk of lost accounts
- **Single-point-of-attack:** risk of compromised accounts

Distributed ARKG aims to address the above problem.

- pk‘ is derived by the sender for a **set** of *initially* **n** proxies.
- sk‘ must be reconstructable by at least **t-out-of-n** proxies.

Desired properties to keep in mind

- Asynchronous communication between sender and each of the proxies during derivation of pk‘.
- Ensuring that sk‘ can be reconstructed once pk‘ is created and the sender goes offline.



Simplified functionality of dARKG

PACY LAB

Proxy i



long-term keys, shares sh_i, dealings d_i:

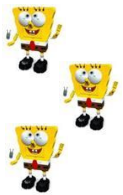
- $\mathrm{LTKGen}(1^\lambda) \rightarrow (dk_i, ek_i)$
- $\mathrm{KGenProx}([ek]_n) \rightarrow (sh_i, d_i)$ with $[ek]_n = (ek_1, \ldots, ek_n)$

DerivePK **Delegator**



verification/aggregation into "group" pk and new dealings f, derivation of pk' and $cred_i$)

- $\mathrm{KGenDeleg}([ek]_n, [d]_n) \rightarrow (pk, [f]_n)$ or $\perp$
- $\mathrm{DerivePK}(pk) \rightarrow (pk^\star, cred)$ backwards compatibility with ARKG
- $\mathrm{ShareDPK}(pk^\star, cred, t, [ek]_n, [f]_n) \rightarrow (pk', [cred]_n)$



Proxies DeriveSK

derivation of secret key share sh'_i and sk' :

- $\mathrm{DeriveSKSh}(dk_i, sh_i, cred_i) \rightarrow sh'_i$
- $\mathrm{DeriveSK}\langle(\mathrm{Proxy}_i)_{1\le i\le t}\rangle \rightarrow sk' / \perp$ (protocol modelled as an algorithm)

Simplified dARKG construction

1PVAKA is a special one-round publicly verifiable asymmetric key agreement.
pp = (AKG.Setup, 1PVAKA.Setup, KEM, H, MAC, KDF, TSS, PKE) with KEM.KGen = AKG.KGen
Proxies have long-term key pairs (dk_i, ek_i) for 1PVAKA, KEM, PKE.





KGenProxy | KGenDeleg

(sh_i, d_i) ←

1PVAKA

← $[ek]_n$

→ d_i

→ $(pk, [f]_n)$ based on verified dealings



DeriveSKSh + DeriveSK

$sk \leftarrow 1PVAKA.CombineSK(dk_i, sh_i, f_i)$
$s_i \leftarrow PKE.Dec(dk_i, c_i)$
$ct \leftarrow TSS.Recons([s]_t)$
$K \leftarrow KEM.Decaps(sk, ct);\ \tau \leftarrow H(K)$
$sk' \leftarrow AKG.BlindSK(sk, \tau)$
return sk'

$cred_i$ ←

DerivePK + ShareDPK

$(K, ct) \leftarrow KEM.Encaps(pk);\ \tau \leftarrow H(K)$
$pk' \leftarrow AKG.BlindPK(pk, \tau)$
$[s]_n \leftarrow TSS.Share(t, ct)$
$cred_i \leftarrow (f_i, c_i := PKE.Enc(ek_i, s_i))$
return pk' and cred $= [cred]_n$

Simplified 1PVAKA construction

1PVAKA can be built from Verifiable (multi-recipient) Encryption, e.g., based on ElGamal and e: $G_1 \times G_2 \to G_T$.
Each proxy i has (dk_i, ek_i) for VE.



$(x_i, g_1^{x_i}) \leftarrow$ AKG.KGen(pp)

$sh_i = x_i \qquad d_i = (g_1^{x_i}, ct_i)$

$ct_i = \{VE.Enc(ek_j, x_i)\}_{j \neq i}$



1PVAKA

$[ek]_n$

d_i

$(pk = g_1^{x_1+\cdots+x_n},\ [f]_n)$ after verifying d_i's



$f_i = \{\text{valid } ct_{j,i}\}$

ElGamal + verification elements

$ct_{i,j}: \ R = g_1^{r}, E = ek_j^{r} g_1^{x_i}, \ U = g_2^{rz}, \ Z = g_2^{z}$

Verification of d_i: $e(R, Z) \stackrel{?}{=} e(g_1, U)$ and $e(E, Z) \stackrel{?}{=} e(ek_j, U)e(g_1^{x_i}, Z)$

1PVAKA. CombineSK(dk_i, sh_i, f_i) : decrypt all x_j from $ct_{j,i}$

$sk = x_1 + \cdots + x_n$

Note: Since x_i are encrypted in exponent we decompose x_i in base B and encrypt blocks individually as in Groth's NI-DKG protocol.

ARKG-related source codes

PACY LAB

- **DL-ARKG**: https://github.com/Yubico/webauthn-recovery-extension/ with fastecdsa, P256 curve.
- **Pairing-based ARKG with ϕ-AKG**: https://gitlab.surrey.ac.uk/sccs/bp-arkg
  - BLS-1, BLS-3, CL (in C with libpbc and libsodium)
  - PS, SPS-EQ, Waters (in Python with bplib)
- **Lattice-based ARKG**: https://gitlab.surrey.ac.uk/sccs/larkg
  - Using FrodoKEM (Level 3 params, fixed matrix **A**) and Kyber (Level 5 params) in Python
  - Kyber as Split KEM functionality, incl. rejection sampling
- **Delegation of WebAuthn credentials**: https://github.com/UoS-SCCS/WebAuthn-Credential-Delegation
  - Proxy signatures based on ARKG: https://github.com/UoS-SCCS/PSUW-Primitive
- **Distributed ARKG**
  - 1PVAKA benchmarks: https://gitlab.com/rv5MDg/jupyter-notebook-darkg with BLS12-381

From account recovery to delegation

PACY LAB

Account recovery via ARKG is like “self-delegation”

- No differentiation between account owner and proxy
- Trusted, single ownership of static and ARKG-derived keys

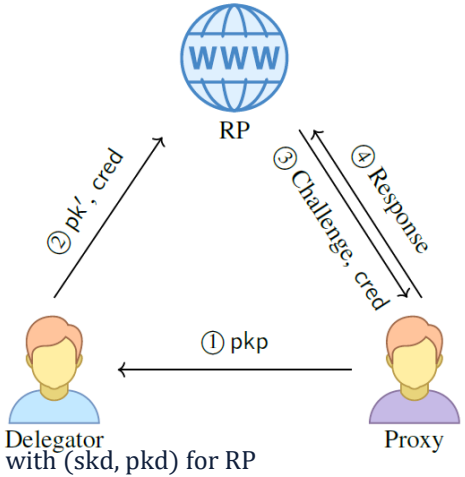
Can we use ARKG to delegate to proxies?

- Things to take into account:
  - RPs to distinguish delegated credentials and control permissions
  - Account owners to revoke / invalidate delegated credentials
  - Online vs offline proxy users, absence of proxy accounts

Remote delegation using ARKG only

PACY LAB

Configurable at RP with proxy being offline.



1. P: (skp, pkp) ← ARKG.KGen(pp), sends pkp to D.
2. D: logs into RP as normal, sends delegated credential (pk‘, cred) ← ARKG.DerivePK(pkp, aux) to RP.
3. RP sends WebAuthn challenge and cred to P.
4. P: sk‘← ARKG.DeriveSK(skp, cred), computes and sends WebAuthn response to RP.

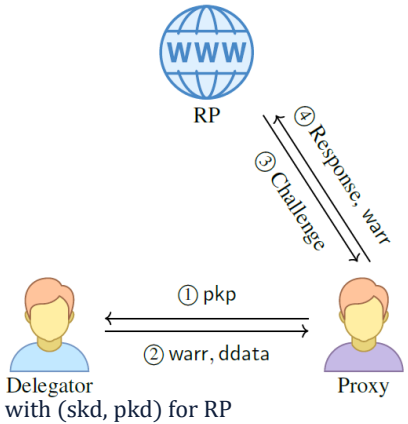
Permissions can be configured at RP with step 2. To revoke access D deletes (pk‘, cred) at RP.

Remote delegation uses Yubico‘s ARKG-based backup approach directly to perform delegation.

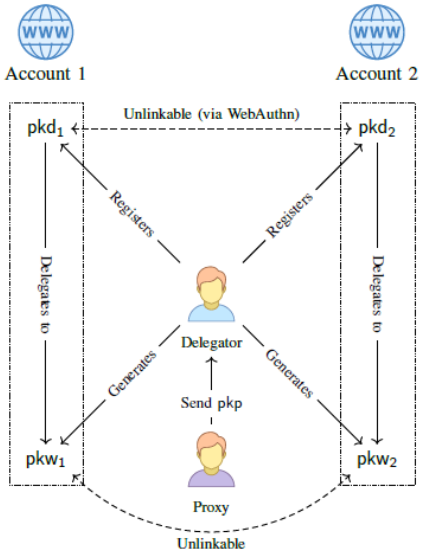
Direct delegation via proxy signatures

PACY LAB

Delegated credentials sent to proxy with RP offline, uses ARKG and a digital signature DS.



1. P: (skp, pkp) ← ARKG.KGen(pp), sends pkp to D. – same as in remote case
2. D: (pkw, cred) ← ARKG.DerivePK(pkp), σ ← DS.Sign(skd, (pkw, aux)), sends (pkw, σ, cred) to P.
3. RP sends WebAuthn challenge to P.
4. P: skw ← ARKG.DeriveSK(skp, cred) to compute WebAuthn response, sends it with (pkw, σ) to RP.



Permissions, etc can be included into aux. To revoke access, D can replace its pkd at RP.

PSUW: This gives a new type of *proxy signatures with unlinkable warrants* (pkw, σ) that do not reveal pkp.

Thank you!

Special thanks to all co-authors

Current PhD students
- Nick Frymann
- Hugo Nartz

Former PhD students
- Daniel Gardham
- Franziskus Kiefer

Yubico
- Emil Lundberg
- Dain Nilson





Mark Manulis

PACY Lab, Research Institute CODE

Universität der Bundeswehr München

mark.manulis@unibw.de

https://www.unibw.de/pacy-en