

pqARKG-H: An extension of pqARKG for Hierarchical Deterministic Keys

Emil Lundberg
Yubico AB
emil@yubico.com

February 3, 2025

Abstract

Hierarchical Deterministic Keys (HDK) is a draft specification using key blinding techniques including ARKG to create hierarchies of keys all derived from a single root secret key, in such a way that only the root secret key needs to be kept in a secure environment and new public keys can be generated without access to the root secret key. However, when ARKG is used in HDK, the original ARKG procedures would require one invocation of `DeriveSK` per HDK layer when exercising derived private keys. This is undesirable in contexts where each invocation would require a distinct user gesture.

To address this we analyze the security of pqARKG-H, an extension of pqARKG for better compatibility with HDK. pqARKG is in turn a post-quantum compatible generalization of ARKG. pqARKG-H enables any number of HDK layers with a single call to the root private key. We demonstrate that pqARKG-H retains the key security properties of pqARKG by a straightforward reduction of the key security experiment.

1 Introduction

Hierarchical Deterministic Keys (HDK) [Dij25] is a draft specification using key blinding techniques including ARKG [LB25] to create hierarchies of keys all derived from a single root secret key, in such a way that only the root secret key needs to be kept in a secure environment and new public keys can be generated without access to the root secret key. However, when ARKG is used in HDK, the original ARKG procedures would require one invocation of `DeriveSK` per HDK layer when exercising derived private keys. This is undesirable in contexts where each invocation would require a distinct user gesture. One such context is the WebAuthn “sign” extension [Lun24] proposed to introduce general-purpose signing capabilities to

the Web Authentication API (WebAuthn) [Bal+25], including signing using private keys derived via ARKG. Since WebAuthn requires a user gesture for any signing operation, HDK operations based on this “sign” extension would also require one user gesture per HDK layer.

The ARKG construction in [LB25] is based upon pqARKG [Wil23], a post-quantum compatible generalization of the original ARKG [Fry+20]. Here we propose and analyze pqARKG-H, an extension of pqARKG for better compatibility with HDK. pqARKG-H enables any number of HDK layers with a single invocation of the `DeriveSK` procedure.

The next section repeats the definition of pqARKG and its `msKS` security experiment. Then we propose the extension pqARKG-H and its corresponding security experiment, and finally demonstrate by a straightforward reduction that pqARKG-H retains the `msKS` property of pqARKG.

2 pqARKG

Here we repeat the definition of pqARKG and its `msKS` security experiment [Wil23]. The instance parameters are a key blinding scheme Δ , a key encapsulation mechanism (KEM) Π and a pseudo-random function PRF outputting blinding factors in the domain of Δ .

pqARKG is defined as the suite of procedures in Figure 2. The `msKS` security experiment for pqARKG is defined in Figure 2.¹

3 pqARKG-H

Here we define our modified pqARKG-H scheme and a corresponding `msKS` security experiment.

A new parameter b is added to the `DerivePK` and `DeriveSK` functions. This b is an additional blinding factor in the key blinding scheme Δ , allowing the ARKG subordinate party (the party generating public keys) to add any number of additional blinding layers on top of the one performed by the ARKG delegating party (the party holding the ARKG private seed). To prevent choosing $b = -\tau$ so that it cancels the blinding factor τ computed in step 2 of `DeriveSK` of pqARKG, this b is also mixed into the PRF arguments to compute τ . This disrupts any algebraic relationship between b and τ , thus preventing the subordinate party from extracting the private seed `sk` by a malicious choice of b .

The new argument mixed into the PRF is however not b directly, but a blinded public key pk_Δ^b incorporating the blinding factor b . This enables the subordinate party to prove to a third party that a public key was generated using pqARKG-H without having to disclose b . This is desirable if the

¹We have corrected a misprint in [Wil23] writing τ in place of k on line 3 of $\mathcal{O}_{\text{sk}'}$, and renamed the variable `ck` to τ in $\mathcal{O}_{\text{pk}'}$.

Setup(λ)

1 : **return** $\text{pp} = (\Delta, \Pi, \text{PRF})$

Check($\text{pp}, \text{sk}', \text{pk}'$)

1 : **return** $\Delta.\text{Check}(\text{sk}', \text{pk}')$

KGen(pp)

1 : $(\text{sk}_\Delta, \text{pk}_\Delta) \leftarrow \Delta.\text{KGen}()$
 2 : $(\text{sk}_\Pi, \text{pk}_\Pi) \leftarrow \Pi.\text{KGen}()$
 3 : **return** $\text{sk} = (\text{sk}_\Delta, \text{sk}_\Pi), \text{pk} = (\text{pk}_\Delta, \text{pk}_\Pi)$

DerivePK($\text{pp}, \text{pk} = (\text{pk}_\Delta, \text{pk}_\Pi), \text{aux}$)

1 : $(c, k) \leftarrow \Pi.\text{Encaps}(\text{pk}_\Pi)$
 2 : $\tau \leftarrow \text{PRF}(k, \text{aux})$
 3 : $\text{pk}' \leftarrow \Delta.\text{BlindPK}(\text{pk}_\Delta, \tau)$
 4 : **return** $\text{pk}', \text{cred} = (c, \text{aux})$

DeriveSK($\text{pp}, \text{sk} = (\text{sk}_\Delta, \text{sk}_\Pi), \text{cred} = (c, \text{aux})$)

1 : $k \leftarrow \Pi.\text{Decaps}(\text{sk}_\Pi, c)$
 2 : $\tau \leftarrow \text{PRF}(k, \text{aux})$
 3 : **return** $\Delta.\text{BlindSK}(\text{sk}_\Delta, \tau)$

Figure 1: Algorithms of the pqARKG scheme [Wil23].

$\text{Exp}_{pqARKG, \mathcal{A}}^{\text{msKS}}(\lambda)$

1 : $\text{pp} \leftarrow \text{Setup}(1^\lambda)$
 2 : $\text{PKList} \leftarrow \emptyset$
 3 : $\text{SKList} \leftarrow \emptyset$
 4 : $(\text{sk}, \text{pk}) \leftarrow \text{KGen}()$
 5 : $(\text{sk}^*, \text{pk}^*, \text{cred}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\text{pk}'}, \mathcal{O}_{\text{sk}'}}(\text{pp}, \text{pk})$
 6 : $\text{sk}' \leftarrow \text{DeriveSK}(\text{pp}, \text{sk}, \text{cred}^*)$
 7 : **return** $\Delta.\text{Check}(\text{sk}^*, \text{pk}^*)$
 8 : $\quad \wedge \Delta.\text{Check}(\text{sk}', \text{pk}^*)$
 9 : $\quad \wedge [\text{cred}^* \notin \text{SKList}]$

$\mathcal{O}_{\text{pk}'}(\text{aux})$

1 : $(c, k) \leftarrow \Pi.\text{Encaps}(\text{pk}_\Pi)$
 2 : $\tau \leftarrow \text{PRF}(k, \text{aux})$
 3 : $\text{pk}' \leftarrow \Delta.\text{BlindPK}(\text{pk}_\Delta, \tau)$
 4 : $\text{PKList} \leftarrow \text{PKList} \cup \{(\text{pk}', (c, \text{aux}))\}$
 5 : **return** $\text{pk}', (c, \text{aux})$

$\mathcal{O}_{\text{sk}'}(c, \text{aux})$

1 : **if** $(\cdot, (c, \text{aux})) \notin \text{PKList}$ **then return** \perp
 2 : $\text{SKList} \leftarrow \text{SKList} \cup \{(c, \text{aux})\}$
 3 : $k \leftarrow \Pi.\text{Decaps}(\text{sk}_\Pi, c)$
 4 : $\tau \leftarrow \text{PRF}(k, \text{aux})$
 5 : **return** $\Delta.\text{BlindSK}(\text{sk}_\Delta, \tau)$

Figure 2: The msKS security experiment for pqARKG [Wil23].

subordinate party does not wish to reveal the relationship with keys from other branches of an HDK tree which might be used for unrelated purposes; knowing b would enable the third party to unblind the derived public key pk' to reveal the root public seed pk_Δ . Instead, the third party may receive k and aux and recompute steps 3-5 of `DerivePK` with $\text{pk}_\Delta = \text{pk}_\Delta^b$ and $b = 1$, the identity blinding factor, and thus be convinced that pk' was generated from the claimed public seed pk_Δ^b .

Finally, the `DeriveSK` function of `pqARKG-H` also receives the blinding key pk_Δ as a new parameter in order to reconstruct the same PRF output as `DerivePK`. This pk_Δ parameter may be eliminated in instantiations where pk_Δ can be computed from sk_Δ .

`pqARKG-H` requires three additional properties of the the key blinding scheme Δ :

1. There exists an *identity blinding factor*, denoted 1 , such that

$$\Delta.\text{BlindPK}(\text{pk}, 1) = \text{pk} \text{ and } \Delta.\text{BlindSK}(\text{sk}, 1) = \text{sk}$$

for all pk and sk .

2. Δ supports *public key unblinding* in addition to private key unblinding: A function $\Delta.\text{UnblindPK}(\text{pk}, b)$ such that

$$\Delta.\text{UnblindPK}(\Delta.\text{BlindPK}(\text{pk}, b), b) = \text{pk}$$

for all pk and b .

3. Δ is *commutative* in the blinding factor: for all pk , sk , a and b ,

$$\Delta.\text{BlindPK}(\Delta.\text{BlindPK}(\text{pk}, a), b) = \Delta.\text{BlindPK}(\Delta.\text{BlindPK}(\text{pk}, b), a)$$

$$\Delta.\text{BlindSK}(\Delta.\text{BlindSK}(\text{sk}, a), b) = \Delta.\text{BlindSK}(\Delta.\text{BlindSK}(\text{sk}, b), a).$$

For example, any construction based on cyclic groups is likely to satisfy these properties.

`pqARKG-H` is defined as the suite of procedures in Figure 2. The operator \parallel denotes binary concatenation, and we assume some well-known encoding is used for pk_Δ^b . Note that if $\Delta.\text{BlindSK}(\text{sk}, b)$ is linear in b , then steps 4-5 of `DeriveSK` may be optimized as "4. **return** $\Delta.\text{BlindSK}(\text{sk}_\Delta, b\tau)$ ".

We also modify the `msKS` security experiment accordingly, resulting in the security experiment $\text{Exp}_{pqARKG-H, \mathcal{B}}^{\text{msKS}}$. The main difference is that the adversary \mathcal{B} also returns the value b^* to be used as the b argument to `DeriveSK`. The public and private key oracles $\mathcal{O}_{\text{pk}'}$ and $\mathcal{O}_{\text{sk}'}$ are also updated to include the b parameter and the additional blinding step. Finally, the check against \mathcal{B} trivially querying $\mathcal{O}_{\text{sk}'}$ for a solution is relaxed to forbid only the exact combination of b^* and cred^* returned as the solution.

Setup: Same as pqARKG.
 Check: Same as pqARKG.
 KGen: Same as pqARKG.

DerivePK(pp, pk = (pk_Δ, pk_Π), b, aux)

```

1 : (c, k) ← Π.Encaps(pkΠ)
2 : pkΔb ← Δ.BlindPK(pkΔ, b)
3 : τ ← PRF(k, pkΔb || aux)
4 : pk' ← Δ.BlindPK(pkΔb, τ)
5 : return pk', cred = (c, aux)

```

DeriveSK(pp, pk_Δ, sk = (sk_Δ, sk_Π), b, cred = (c, aux))

```

1 : k ← Π.Decaps(skΠ, c)
2 : pkΔb ← Δ.BlindPK(pkΔ, b)
3 : τ ← PRF(k, pkΔb || aux)
4 : skΔb ← Δ.BlindSK(skΔ, b)
5 : return Δ.BlindSK(skΔb, τ)

```

Figure 3: Algorithms of the pqARKG-H scheme. The Setup, Check and KGen algorithms are unchanged from pqARKG.

Exp_{pqARKG-H, B}^{msKS}(λ)

```

1 : pp ← Setup(1λ)
2 : PKList ← ∅
3 : SKList ← ∅
4 : (sk, pk) ← KGen()
5 : (sk*, pk*, b*, cred*) ← BOpk', Osk'(pp, pk)
6 : sk' ← DeriveSK(pp, pkΔ, sk, b*, cred*)
7 : return Δ.Check(sk*, pk*)
8 :   ∧ Δ.Check(sk', pk*)
9 :   ∧ [(b*, cred*) ∉ SKList]

```

O_{pk'}(b, aux)

```

1 : (pk', cred) ← DerivePK(pp, pk, b, aux)
2 : PKList ← PKList ∪ {(pk', cred)}
3 : return pk', cred

```

O_{sk'}(b, c, aux)

```

1 : if (·, (c, aux)) ∉ PKList then return ⊥
2 : SKList ← SKList ∪ {(c, aux)}
3 : return DeriveSK(pp, pkΔ, sk, b, (c, aux))

```

Figure 4: The msKS security experiment for pqARKG-H.

$\mathcal{A}^{\mathcal{O}_{pk'}, \mathcal{O}_{sk'}}(pp = (\Delta, \Pi, \text{PRF}), pk = (pk_\Delta, pk_\Pi))$	$\overline{\mathcal{O}_{pk'}}(b, aux)$
1 : $(sk^*, pk^*, \bar{b}^*, \overline{cred^*}) \leftarrow_{\$} \mathcal{B}^{\mathcal{O}_{pk'}, \mathcal{O}_{sk'}}(pp, pk)$	1 : $pk_\Delta^b \leftarrow \Delta.\text{BlindPK}(pk_\Delta, b)$
2 : $sk^* \leftarrow \Delta.\text{UnblindSK}(\overline{sk^*}, \bar{b}^*)$	2 : $(pk', (c, \cdot)) \leftarrow_{\$} \mathcal{O}_{pk'}(pk_\Delta^b \ aux)$
3 : $pk^* \leftarrow \Delta.\text{UnblindPK}(\overline{pk^*}, \bar{b}^*)$	3 : $\overline{pk'} \leftarrow \Delta.\text{BlindPK}(pk', b)$
4 : $pk_\Delta^{b^*} \leftarrow \Delta.\text{BlindPK}(pk_\Delta, \bar{b}^*)$	4 : return $(\overline{pk'}, (c, aux))$
5 : return $(sk^*, pk^*, (\bar{c}^*, pk_\Delta^{b^*} \ \overline{aux^*}))$	

$\overline{\mathcal{O}_{sk'}}(b, c, aux)$
1 : $pk_\Delta^b \leftarrow \Delta.\text{BlindPK}(pk_\Delta, b)$
2 : $sk' \leftarrow \mathcal{O}_{sk'}(c, pk_\Delta^b \ aux)$
3 : return $\Delta.\text{BlindSK}(sk', b)$

Figure 5: Reduction of $\text{Exp}_{pqARKG, \mathcal{A}}^{\text{msKS}}$ to $\text{Exp}_{pqARKG-H, \mathcal{B}}^{\text{msKS}}$.

4 Reduction of pqARKG to pqARKG-H in msKS security experiment

We now show that pqARKG-H is msKS secure by showing that an adversary \mathcal{B} that defeats $\text{Exp}_{pqARKG-H, \mathcal{B}}^{\text{msKS}}$ also defeats $\text{Exp}_{pqARKG, \mathcal{A}}^{\text{msKS}}$. Given such an adversary \mathcal{B} , we construct an adversary \mathcal{A} that defeats $\text{Exp}_{pqARKG, \mathcal{A}}^{\text{msKS}}$ as defined in Figure 4.

When invoked, this adversary \mathcal{A} simply invokes the given adversary \mathcal{B} with its own challenge. The $\mathcal{O}_{pk'}$ and $\mathcal{O}_{sk'}$ oracles are adapted for \mathcal{B} by adding the pk_Δ^b prefix to aux and performing the additional blinding with the b argument, thus \mathcal{A} faithfully simulates $\text{Exp}_{pqARKG-H, \mathcal{B}}^{\text{msKS}}$ for \mathcal{B} . The aux^* returned to the challenger is adapted with the $pk_\Delta^{b^*}$ prefix computed using the blinding factor $\bar{b}^* = b^*$ returned by \mathcal{B} , thus pqARKG-H produces the same τ on line 3 of its `DerivePK` and `DeriveSK` functions as pqARKG does on line 2 of its `DerivePK` and `DeriveSK` functions.

We see that \mathcal{A} passes the conditions on lines 7-9 of $\text{Exp}_{pqARKG, \mathcal{A}}^{\text{msKS}}$:

- $\Delta.\text{Check}(sk^*, pk^*)$ succeeds because:
 $sk^* = \Delta.\text{UnblindSK}(\overline{sk^*}, \bar{b}^*)$ and $pk^* = \Delta.\text{UnblindPK}(\overline{pk^*}, \bar{b}^*)$. Therefore by the definitions of unblinding and unique blinding, \mathcal{A} passes its $\Delta.\text{Check}(sk^*, pk^*)$ condition precisely when \mathcal{B} does.
- $\Delta.\text{Check}(sk', pk^*)$ succeeds because:
First, observe that $c^* = \bar{c}^*$ and $aux^* = pk_\Delta^{b^*} \| \overline{aux^*}$, and therefore
 $\tau = \text{PRF}(\Pi.\text{Decaps}(sk_\Pi, c^*), aux^*) = \text{PRF}(\Pi.\text{Decaps}(sk_\Pi, \bar{c}^*), pk_\Delta^{b^*} \| \overline{aux^*})$

so DeriveSK on line 6 of $\text{Exp}_{pqARKG, \mathcal{A}}^{\text{msKS}}$ computes the same τ as on line 6 of $\text{Exp}_{pqARKG-H, \mathcal{B}}^{\text{msKS}}$.

The above gives

$$\begin{aligned} \text{sk}' &= \text{DeriveSK}(\text{pp}, \text{sk}, \text{cred}^*) = \\ &= \Delta.\text{BlindSK}(\text{sk}_\Delta, \text{PRF}(\Pi.\text{Decaps}(\text{sk}_\Pi, c^*), \text{aux}^*)) \\ &= \Delta.\text{BlindSK}(\text{sk}_\Delta, \tau) \end{aligned}$$

where $\text{cred}^* = (c^*, \text{aux}^*)$.

Let $\overline{\text{sk}'}$ be an alias of the sk' on line 6 of $\text{Exp}_{pqARKG-H, \mathcal{B}}^{\text{msKS}}$. Then

$$\begin{aligned} \overline{\text{sk}'} &= \text{DeriveSK}(\text{pp}, \text{pk}_\Delta, \text{sk}, \overline{b^*}, \overline{\text{cred}^*}) = \\ &= \Delta.\text{BlindSK}(\Delta.\text{BlindSK}(\text{sk}_\Delta, \overline{b^*}), \tau) = \\ &= \Delta.\text{BlindSK}(\Delta.\text{BlindSK}(\text{sk}_\Delta, \tau), \overline{b^*}) = \\ &= \Delta.\text{BlindSK}(\text{sk}', \overline{b^*}) \\ &\implies \text{sk}' = \Delta.\text{UnblindSK}(\overline{\text{sk}'}, \overline{b^*}) \end{aligned}$$

so by the definitions of unblinding and unique blinding, the $\Delta.\text{Check}(\text{sk}', \text{pk}^*)$ condition in $\text{Exp}_{pqARKG-H, \mathcal{B}}^{\text{msKS}}$ is equivalent to

$$\begin{aligned} &\Delta.\text{Check}(\Delta.\text{BlindSK}(\text{sk}', \overline{b^*}), \overline{\text{pk}^*}) = \\ &= \Delta.\text{Check}(\Delta.\text{BlindSK}(\text{sk}', \overline{b^*}), \Delta.\text{BlindPK}(\Delta.\text{UnblindPK}(\overline{\text{pk}^*}, \overline{b^*}), \overline{b^*})) = \\ &= \Delta.\text{Check}(\Delta.\text{BlindSK}(\text{sk}', \overline{b^*}), \Delta.\text{BlindPK}(\text{pk}^*, \overline{b^*})) = \\ &= \Delta.\text{Check}(\text{sk}', \text{pk}^*) \end{aligned}$$

which is precisely the condition on line 8 of $\text{Exp}_{pqARKG, \mathcal{A}}^{\text{msKS}}$. Therefore \mathcal{A} passes its $\Delta.\text{Check}(\text{sk}', \text{pk}^*)$ condition precisely when \mathcal{B} does.

- $(c^*, \text{aux}^*) \notin \text{SKList}$ succeeds because:

SKList is appended to only when \mathcal{A} invokes $\mathcal{O}_{\text{sk}'}$, which it does only when \mathcal{B} invokes $\overline{\mathcal{O}_{\text{sk}'}}$. Therefore if \mathcal{B} does not invoke $\overline{\mathcal{O}_{\text{sk}'}}$ with arguments (b^*, c^*, aux^*) , then \mathcal{A} also does not invoke $\mathcal{O}_{\text{sk}'}$ with arguments (c^*, aux^*) . This is the case when \mathcal{B} wins its game, therefore \mathcal{A} passes this condition when \mathcal{B} does.

In conclusion, we see that \mathcal{A} wins its game precisely when \mathcal{B} wins its game, therefore the advantages are equal:

$$\text{Adv}_{pqARKG-H, \mathcal{B}}^{\text{msKS}}(\lambda) = \text{Adv}_{pqARKG, \mathcal{A}}^{\text{msKS}}(\lambda)$$

Thus we conclude that pqARKG-H retains the msKS property of pqARKG .

5 Acknowledgements

We would like to thank Sander Dijkhuis for proposing the construction with mixing pk_Δ^b into the PRF instead of b , and for helping find a successful approach for the security reduction.

References

- [Bal+25] Dirk Balfanz et al. *Web Authentication: An API for accessing Public Key Credentials*. 2025. URL: <https://www.w3.org/TR/2025/WD-webauthn-3-20250127/>. Level 3 working draft.
- [Dij25] Sander Dijkhuis. *Hierarchical Deterministic Keys*. 2025. URL: <https://www.ietf.org/archive/id/draft-dijkhuis-cfrg-hdkeys-06.html>. Version 06.
- [Fry+20] Nick Frymann et al. “Asynchronous Remote Key Generation: An Analysis of Yubico’s Proposal for W3C WebAuthn”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 939–954. ISBN: 9781450370899. DOI: 10.1145/3372297.3417292. URL: <https://doi.org/10.1145/3372297.3417292>.
- [LB25] Emil Lundberg and John Bradley. *The Asynchronous Remote Key Generation (ARKG) algorithm*. 2025. URL: <https://www.ietf.org/archive/id/draft-bradleylundberg-cfrg-arkg-03.html>. Version 03.
- [Lun24] Emil Lundberg. *Add “sign” extension*. 2024. URL: <https://github.com/w3c/webauthn/pull/2078>. W3C Web Authentication issue tracker, accessed 2025-01-31.
- [Wil23] Spencer MacLaren Wilson. *Post-Quantum Account Recovery for Passwordless Authentication*. 2023. URL: <https://uwaterloo.ca/items/d1f73f71-e3b2-438c-b261-11632becdbb2>.