A Lightweight Hybrid Signcryption Scheme for Smart Devices

Lanlan Pan (潘蓝兰)

abbypan@gmail.com

SIN'24 CONF

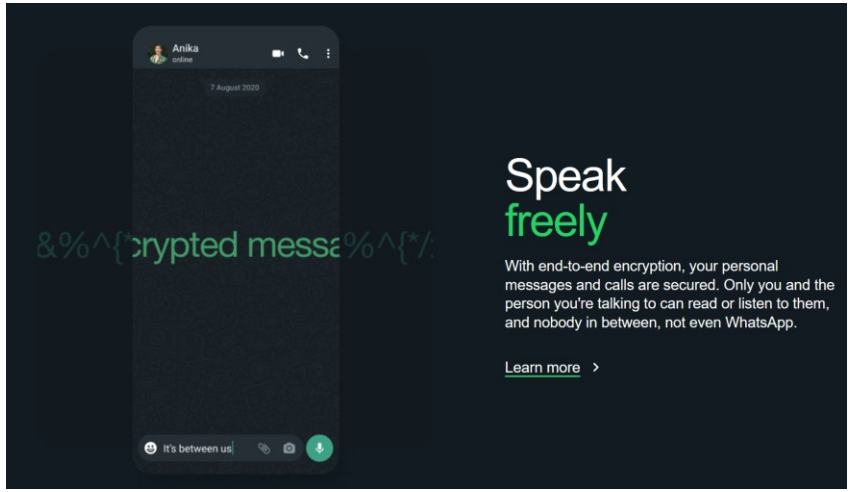
IEEE Advancing Technology for Humanity

- Introduction
- Related Work
- Our Scheme
- Conclusion

Introduction

- A vast number of messages are sent through smart devices every day.
- Devices could make offline and online communication by using message applications such as iMessage and WhatsApp.

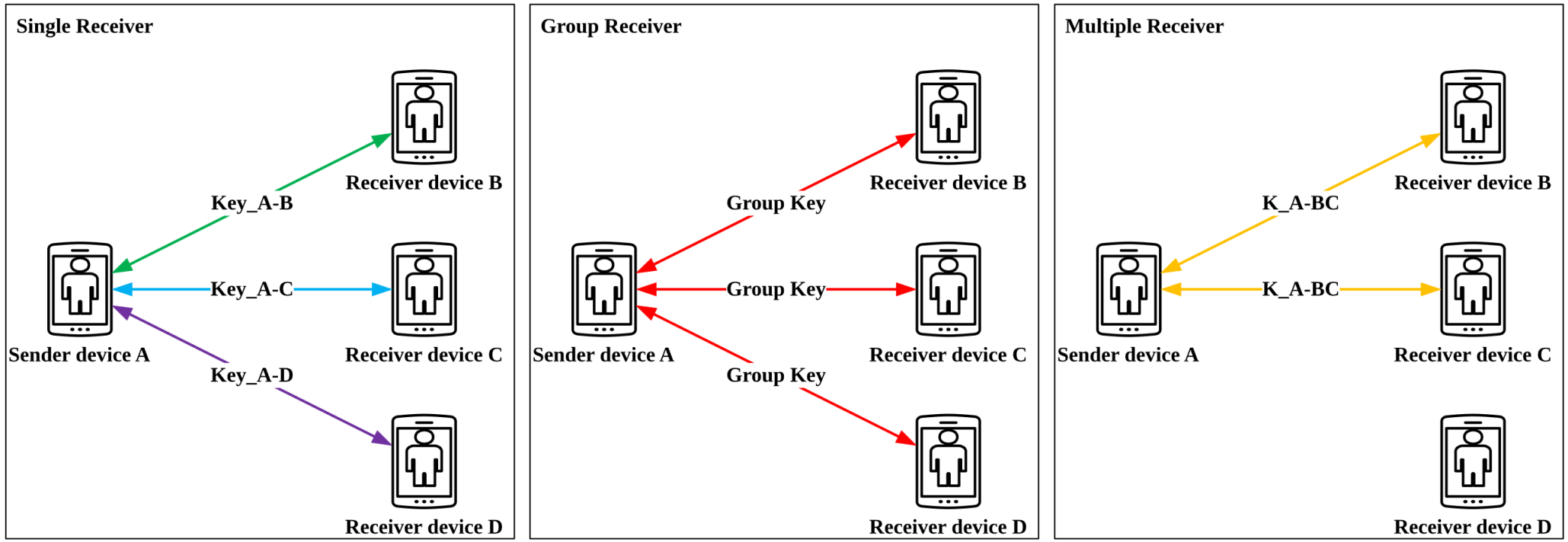
https://support.apple.com/messages



https://www.whatsapp.com/

Multiple-receiver

- The sender device no longer only communicates with single-receiver device but also with group-receiver devices.
- The sender device may send messages to multiple-receiver devices selected from the group device set.



Challenges

- Support multiple-receiver scenario.
  - Make individual message encryption for the selected multiple-receiver devices rather than just reusing the group key.
- Support offline and online communications.
  - The key exchange protocol of online communication can usually support many advanced security properties, such as key-compromise impersonation (KCI) resistance, forward secrecy, post-compromise security, etc.
  - The key exchange protocol of offline communication may make some security trade-offs.
  - It is valuable to support offline and online communications and be compatible with existing message applications.
- Computational and communication cost optimization.
  - Design a lightweight signcryption scheme to make computational cost optimization for the large message.
  - The sender device should avoid re-encrypting plaintext messages for multiple-receiver devices.
  - Since each receiver device only decrypts its ciphertext message once, making the encryption cost lower than decryption is an efficient option for reducing the sender device’s computational and communication costs.

Related Work (Single-receiver): iMessage-RSA

- The plaintext is encrypted by a symmetric key; the symmetric key is encrypted by RSA-OAEP; the whole ciphertext is signed with ECDSA.
- Garman et al. [5] conducted a thorough analysis of iMessage to determine the security of the protocol against a variety of attacks.

iMessage-Enc(Y_b, x_a, m)

1. $L \leftarrow$ random 88-bit value
2. $h \leftarrow \text{HMAC}(L, Y_a \,||\, Y_b \,||\, m)[1..40]$
3. $k \leftarrow L||h$
4. $c_1 \leftarrow \text{AES-CTR.Enc}(k, m)$
5. $c_2 \leftarrow \text{RSA-OAEP.Enc}(Y_b, k)$
6. $dgst \leftarrow SHA1(c_1 \,||\, c_2)$
7. $s \leftarrow \text{ECDSA.Sign}(x_a, dgst)$
8. return (c_1, c_2, s).

iMessage-Dec(Y_a, x_b, c_1, c_2, s)

1. $dgst \leftarrow SHA1(c_1 \,||\, c_2)$
2. $v \leftarrow \text{ECDSA.Verify}(Y_a, dgst, s)$; if $v = 0$, return.
3. $k \leftarrow \text{RSA-OAEP.Dec}(x_b, c_2)$
4. $m \leftarrow \text{AES-CTR.Dec}(k, c_1)$
5. $L \leftarrow k[1..88]$
6. $h \leftarrow k[89..128]$
7. $h' \leftarrow \text{HMAC}(L, Y_a \,||\, Y_b \,||\, m)[1..40]$
8. if $h' \neq h$, return.
9. return m.

Related Work (Single-receiver): ECIES-Sig

- The plaintext is encrypted by ECIES; the whole ciphertext is signed with ECDSA.
- In iOS 13 or later and iPadOS 13.1 or later, Apple devices may use ECIES-Sig instead of iMessage-RSA.

ECIES- Sig- Encryption($\mathbf{Y_b}, \mathbf{x_a}, \mathbf{m}$)

1. $c \leftarrow \text{ECIES. Enc}(Y_b, \text{m})$
2. $\text{dgst} \leftarrow SHA256(c)$
3. $\text{s} \leftarrow \text{ECDSA. Sign}(x_a, \text{dgst})$
4. return (c, s).

ECIES- Sig- Decryption($\mathbf{Y_a}, \mathbf{x_b}, \mathbf{c}, \mathbf{s}$)

1. $\text{dgst} \leftarrow SHA256(c)$
2. $v \leftarrow \text{ECDSA. Verify}(Y_a, \text{dgst}, \text{s})$; if $v = 0$, return.
3. $\text{m} \leftarrow \text{ECIES. Dec}(x_b, c)$
4. return m.

Related Work (Single-receiver): HPKE

- HPKE provided confidentiality of the sender's messages against chosen ciphertext attacks, proof of sender origin for Pre-Shared Key (PSK), Auth, and AuthPSK modes.
- Alwen et al. [7] proved security of the authenticated key encapsulation mechanism underlying HPKE based on the gap Diffie-Hellman assumption.

HPKEAuth-Enc(Y_b, x_a, m, seq, mode, info, psk, psk_id)

1. $(x_e, Y_e) \leftarrow$ generate key pair randomly
2. dh $\leftarrow$ DH(x_e, Y_b) || DH(x_a, Y_b)
3. $kem\_context \leftarrow Y_e$ || Y_b|| Y_a
4. $share\_secret \leftarrow$ ExtractAndExpand(dh, $kem\_context$)
5. $psk\_id\_hash \leftarrow$ LabeledExtract("", "psk_id_hash", psk_id)
6. $info\_hash \leftarrow$ LabeledExtract("", "info_hash", info)
7. $key\_schedule\_ctx \leftarrow$ mode || psk_id_hash || info_hash
8. $secret \leftarrow$ LabeledExtract($share\_secret$, "secret", psk)
9. $key \leftarrow$ LabeledExpand($secret$, "key", $key\_schedule\_ctx$)
10. $base\_nonce \leftarrow$ LabeledExpand($secret$, "base_nonce", $key\_schedule\_ctx$)
11. $nonce \leftarrow$ XOR$(base_nonce, seq)$
12. $(c, t) \leftarrow Seal(key, nonce, m)$
13. return (Y_e, c, t).

HPKEAuth-Dec(Y_a, x_b, Y_e, c, t, seq, mode, info, psk, psk_id)

1. dh $\leftarrow$ DH(x_b, Y_e) || DH(x_b, Y_a)
2. $kem\_context \leftarrow Y_e$ || Y_b|| Y_a
3. $share\_secret \leftarrow$ ExtractAndExpand(dh, $kem\_context$)
4. $psk\_id\_hash \leftarrow$ LabeledExtract("", "psk_id_hash", psk_id)
5. $info\_hash \leftarrow$ LabeledExtract("", "info_hash", info)
6. $key\_schedule\_ctx \leftarrow$ mode || psk_id_hash || info_hash
7. $secret \leftarrow$ LabeledExtract($share\_secret$, "secret", psk)
8. $key \leftarrow$ LabeledExpand($secret$, "key", $key\_schedule\_ctx$)
9. $base\_nonce \leftarrow$ LabeledExpand($secret$, "base_nonce", $key\_schedule\_ctx$)
10. $nonce \leftarrow$ XOR$(base_nonce, seq)$
11. $m \leftarrow$ UnSeal$(key, nonce, c, t)$
12. return m.

Related Work (Group-receiver): WhatsApp

- WhatsApp uses the Signal protocol to establish a communication channel for messaging between two devices.
- Each device generates a symmetric chain key (k_{chain}) to derive a message key for encrypting only its messages to the group.

The first time a WhatsApp group member sends a message to a group:

1. The sender generates a random 32-byte `Chain Key`.
2. The sender generates a random Curve25519 `Signature Key` key pair.
3. The sender combines the 32-byte `Chain Key` and the public key from the `Signature Key` into a `Sender Key` message.
4. The sender individually encrypts the `Sender Key` to each member of the group, using the pairwise messaging protocol explained previously.

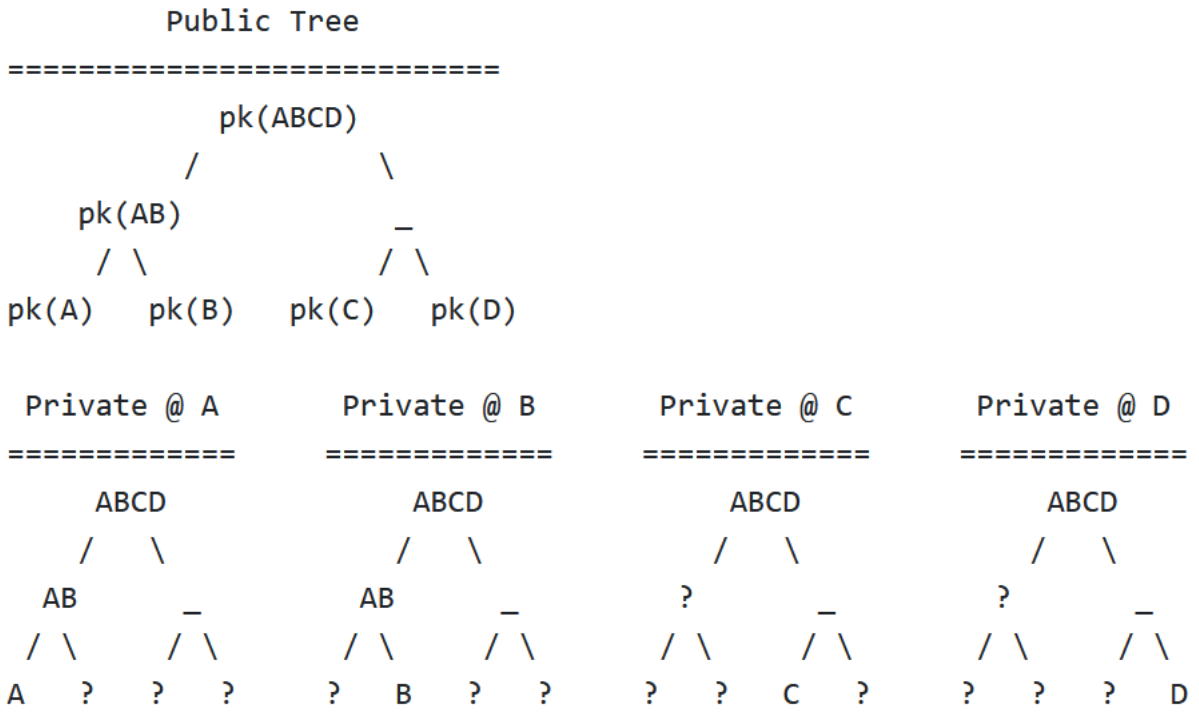
For all subsequent messages to the group:

1. The sender derives a `Message Key` from the `Chain Key`, and updates the `Chain Key`.
2. The sender encrypts the message using `AES256` in CBC mode.
3. The sender signs the ciphertext using the `Signature Key`.
4. The sender transmits the single ciphertext message to the server, which does server-side fan-out to all group participants.

https://faq.whatsapp.com/820124435853543

Related Work (Group-receiver): MLS

- MLS provides asynchronous E2E group management by allowing group members to agree on a fresh independent symmetric key after every change to the group's state.
- Any change to a group's state initiates a new epoch key (k_{epoch}), which can be derived by all group members in the epoch.



https://datatracker.ietf.org/doc/rfc9420/

Our Scheme

- Based on Zheng's signcryption scheme [10], we propose a lightweight hybrid signcryption scheme, supports single-receiver scenarios, and is specifically tailored for multiple-receiver scenarios.

Our Scheme: Context Key

- The scheme includes a context key associated with existing offline and online communication channels.

- The sender and receiver devices negotiate a context key (k_{ctx}) associated with the communication channel.
  - The sender and receiver devices could use some out-of-band provisioned key as k_{ctx}, for example, the pre-shared key (psk) of HPKE [6].
  - In online communication, they can reuse the derived symmetric key of the key exchange phase as kctx, for example, the chain key (k_{chain}) of WhatsApp [2], and the epoch key (k_{epoch}) of MLS [8].

- The context key is optional.

Our Scheme: Single-receiver Communication

- The scheme reduces message encrypting computational time on the sender side by employing only one elliptic curve point multiplication (rY_b).
- The scheme reduces payload size by incorporating authenticated tags (t) in message signature computation.

SINGLE-RECEIVER: ENCRYPTION

$HyperSC\text{-}Enc(x_a, Y_a, id_a, Y_b, id_b, k_{ctx}, m)$
{
$r \leftarrow$ secure random from $[1, ..., q-1]$
$Z \leftarrow rY_b$
$ikm \leftarrow HASH(Z)$

$ctx \leftarrow Y_a||Y_b||id_a||id_b||k_{ctx}$
$klen \leftarrow$ key length of the AEAD cipher
$k \leftarrow HKDF(ikm, ctx, \text{'}key\text{'}, klen)$
$iv \leftarrow$ secure random initialization vector for the AEAD cipher
$(c, t) \leftarrow AEAD\text{-}Enc(k, iv, m)$
$s \leftarrow \frac{r}{t+x_a} \ mod \ q$

$return \ (iv, c, t, s)$
}

SINGLE-RECEIVER: DECRYPTION

$HyperSC\text{-}Dec(x_b, Y_b, id_b, Y_a, id_a, k_{ctx}, iv, c, t, s)$
{
$Z' \leftarrow sx_b(tP + Y_a)$
$ikm \leftarrow HASH(Z')$

$ctx \leftarrow Y_a||Y_b||id_a||id_b||k_{ctx}$
$klen \leftarrow$ key length of the AEAD cipher
$k \leftarrow HKDF(ikm, ctx, \text{'}key\text{'}, klen)$
$m \leftarrow AEAD\text{-}Dec(k, iv, c, t)$

$return \ m$
}

Our Scheme: Multiple-receiver Communication

- The sender device uses km to encrypt plaintext message (m) only once, and makes HyperSC-Enc on k_m for each receiver device.
- The scheme preserves an individual message key between the sender device and selected multiple-receiver devices, not just using the group key.

MULTIPLE-RECEIVER: ENCRYPTION

$HyperSCMulti\text{-}Enc(x_a, Y_a, id_a, \{(Y_j, id_j) \mid 1 \leq j \leq N\}, k_{ctx}, m)$
{
  $iv \leftarrow$ secure random initialization vector for the AEAD cipher
  $k_m \leftarrow$ secure random key for the AEAD cipher
  $(c, t) \leftarrow AEAD\text{-}Enc(k_m, iv, m)$

  for each $j\text{-}th$ receiver device:
    $(iv_j, c_j, t_j, s_j) \leftarrow HyperSC\text{-}Enc(x_a, Y_a, id_a, Y_j, id_j, k_{ctx}, k_m)$

  $return\ (iv, c, t, \{(iv_j, c_j, t_j, s_j) \mid 1 \leq j \leq N\})$
}

MULTIPLE-RECEIVER: DECRYPTION

$HyperSCMulti\text{-}Dec(x_j, Y_j, id_j, Y_a, id_a, k_{ctx}, iv, c, t, iv_j, c_j, t_j, s_j)$
{
  $k_m \leftarrow HyperSC\text{-}Dec(x_j, Y_j, id_j, Y_a, id_a, k_{ctx}, iv_j, c_j, t_j, s_j)$
  $m \leftarrow AEAD\text{-}Dec(k_m, iv, c, t)$
  $return\ m$
}

Security Analysis

We use ProVerif [15] to perform the formal analysis:

- Confidentiality
- Integrity
- Authentication
- KCI: The receiver device can ensure the authentication of the sender device even when the receiver device's private key is leak.

PROVERIF CODE

```
let processSend(a : exponent, principalA : G) =
  let (selfA : G) = exp(P, a) in
  new r : exponent;
  let (Z : G) = dis_exp2(principalA, r) in
  let (k : key) = hkdf(hash(Z), concat(selfA, principalA, getid(selfA), getid(principalA), kCTX)) in
  event termA(a, principalA, k);

  let (ciphertext : bitstring) = symenc(k, secretMsg) in
  let (tag : exponent) = symtag(k, ciphertext) in
  let (sig : exponent) = ediv(r, eadd(tag, a)) in
  event ASend(arincipalA, secretMsg);
  event ASendKM(arincipalA, k, secretMsg);
  out(ch, (ciphertext, tag, sig))
  .

let processRecv(b : exponent, principalB : G) =
  let (selfB : G) = exp(P, b) in
  in(ch, (ciphertext : bitstring, tag : exponent, sig : exponent));
  let (tPYa : G) = dis_add(exp(P, tag), principalB) in
  let (Z : G) = dis_exp(dis_exp(tPYa, sig), b) in
  let (k : key) = hkdf(hash(Z), concat(principalB, selfB, getid(principalB), getid(selfB), kCTX)) in
  event termB(b, principalB, k);

  let (msg : bitstring) = symdec(k, ciphertext) in
  if symtag(k, ciphertext) = tag then (
    event BRecv(brincipalB, msg);
    event BRecvKM(brincipalB, k, msg);
    event BRecvMsg(msg)
  )
  .

process
  out(ch, (b, kCTX, exp(P, a), exp(P, b)));
  (!processSend(a, exp(P, b))|!processRecv(b, exp(P, a)))
```

PROVERIF RESULT

```
Verification summary:
Query not attacker(secretMsg[]) is true.
Query not event(BRecvMsg(secretMsg[])) is false.
Query event(BRecv(x_b, exp(P, x_a), m)) == event(ASend(x_a, exp(P, x_b), m)) is true.
Query event(BRecvKM(x_b, exp(P, x_a), k_2, m)) == event(ASendKM(x_a, exp(P, x_b), k_2, m)) is true.
```

Scheme Comparison

In the single-receiver scenario:

- Same with iMessage-RSA, ECIES-Sig, and HPKE, our scheme supports one-shot message encryption and decryption, satisfies the confidentiality, authentication, and integrity, and does not provide forward secrecy.
- HPKE and our scheme mitigate malleability problems by including all public keys in the context of the key schedule.
- ECIES-Sig and our scheme are resistant to KCI attacks. HPKE is vulnerable to KCI attacks.

In the multiple-receiver scenario:

- Our scheme could use the derived symmetric group key as the context key (k_{ctx}) from existing communication channels such as WhatsApp and MLS.
- Through the context key, our scheme could inherit the advance security properties of existing communication channels, such as forward secrecy, anti-replay, post-compromisesecurity, etc.

Evaluation: Parameters

- We performed the evaluation on a 64-bit Allwinner H616 ARM Cortex-A53 microprocessor (4-Core, 4-Thread, 1.5 GHz).

PARAMETERS SELECTED FOR EVALUATION

Scheme	Crypto	HASH	KDF	Cipher
iMessage-RSA	P-256, RSA1024	SHA1	/	AES-CTR
ECIES-Sig	P-256	SHA256	HKDF	AES-CTR
HPKE	P-256	SHA256	HKDF	AES-GCM
Our Scheme	P-256	SHA256	HKDF	AES-GCM

Evaluation: Single-Receiver

- Our scheme contains the minimum payload size of the four schemes and is approximate with HPKE.
  - Note that the payload size difference in these four schemes will become negligible when the plaintext message length increases.
- Our scheme consumes the minimum CPU time of the four schemes.
  - When |m| ⩽ 1M B, HPKE consumes the maximum CPU time in the four schemes because it calls more Diffie-Hellman, HASH, and KDF calculations.
  - When |m| ⩾ 10M B, iMessage-RSA and ECIES-Sig consume more CPU time than HPKE and our scheme because they make heavy HASH calculations on the plaintext message.
  - The average CPU time of our scheme is approximate with HPKE when the plaintext message length increases.
  - For big attachment scenario, iMessage only relays the messages up to 4 or 16 KB in size, the evaluation matches the |m| ⩽ 1M B.

APNs can only relay messages up to 4 or 16 KB in size, depending on the iOS or iPadOS version. If the message text is too long or if an attachment such as a photo is included, the attachment is encrypted using AES in CTR mode with a randomly generated 256-bit key and uploaded to iCloud. The AES key for the attachment, its Uniform Resource Identifier (URI), and an SHA-1 hash of its encrypted form are then sent to the recipient as the contents of an iMessage, with their confidentiality and integrity protected through normal iMessage encryption, as shown in the following diagram.

https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf

SINGLE-RECEIVER: PAYLOAD SIZE (BYTES)

Options $\|m\|$	Plaintext Message Length ($\|m\|$) 10KB	100KB	1MB	10MB	50MB
iMessage-RSA	10455	102615	1048791	10485975	52429015
ECIES-Sig	10392	102552	1048728	10485912	52428952
HPKE	10321	102481	1048657	10485841	52428881
Our Scheme	10304	102464	1048640	10485824	52428864

Table X
SINGLE-RECEIVER MESSAGE ENCRYPTION: COMPUTATIONAL TIME (MILLISECONDS)

Options $\|m\|$	Plaintext Message Length ($\|m\|$) 10KB	100KB	1MB	10MB	50MB
iMessage-RSA	13.613	14.407	20.305	77.479	333.248
ECIES-Sig	15.392	15.853	20.149	62.044	247.819
HPKE	26.383	26.955	29.610	54.772	167.067
Our Scheme	4.793	5.163	7.859	33.066	144.651

Table XI
SINGLE-RECEIVER MESSAGE DECRYPTION: COMPUTATIONAL TIME (MILLISECONDS)

Options $\|m\|$	Plaintext Message Length ($\|m\|$) 10KB	100KB	1MB	10MB	50MB
iMessage-RSA	11.547	12.291	18.058	76.369	333.545
ECIES-Sig	10.405	10.871	14.980	56.679	242.211
HPKE	25.890	26.172	28.905	55.098	172.379
Our Scheme	4.927	5.322	8.051	34.292	151.071

Evaluation: Multiple-Receiver

- Our scheme contains the minimum payload size of the four schemes and is approximate with HPKE.
  - Note that the payload size difference in these four schemes will become negligible when the plaintext message length increases.
- Our scheme consumes the minimum CPU time of the four schemes.
  - The whole CPU time for the sender device is (menctime + N ∗ kenctime).
  - The whole CPU time for each receiver device is (mdectime + kdectime).
  - Our scheme consumes the minimum CPU time in k_m of the four schemes.
  - Since the encryption cost of our scheme is lower than that of decryption, our scheme could save more CPU time when the number of receiver devices (N) increases.

MULTIPLE-RECEIVER: PAYLOAD SIZE (BYTES)

Options	Plaintext Message Length ($\|m\|$)				
$\|m\|$	**10KB**	**100KB**	**1MB**	**10MB**	**50MB**
iMessage-RSA	10487	102647	1048823	10486007	52429047
ECIES-Sig	10424	102584	1048759	10485944	52428984
HPKE	10353	102513	1048689	10485873	52428913
Our Scheme	10336	102496	1048672	10485855	52428896

Table XIII
MULTIPLE-RECEIVER MESSAGE ENCRYPTION/DECRYPTION: COMPUTATIONAL TIME (MILLISECONDS)

Options	Plaintext Message Length ($\|m\|$)				
$\|m\|$	**10KB**	**100KB**	**1MB**	**10MB**	**50MB**
$menctime$	0.143	0.447	3.112	29.136	144.537
$mdectime$	0.065	0.356	3.223	29.812	147.391

Table XIV
MULTIPLE-RECEIVER SYMMETRIC KEY ENCRYPTION: COMPUTATIONAL TIME FOR EACH RECEIVER DEVICE (MILLISECONDS)

Options	Number of Receiver Devices (N)				
N	**10**	**50**	**100**	**200**	**500**
iMessage-RSA	13.138	13.217	13.250	13.310	13.495
ECIES-Sig	15.423	15.626	15.675	15.743	15.936
HPKE	27.009	27.198	27.218	27.222	27.220
Our Scheme	4.746	4.790	4.795	4.797	4.798

Conclusion

- Our Work
  - We propose a lightweight hybrid signcryption scheme to optimize for single-receiver and multiple-receiver scenarios.
  - Our scheme incorporates a hybrid signcryption one-shot message scheme, which offers several security features such as confidentiality, authentication, integrity, and KCI resistance.
  - Evaluation results demonstrate that the proposed scheme optimizes computational time and reduces payload size compared to existing schemes.
- Limitation
  - We focus on optimizing message encryption on the data layer for single-receiver and multiple-receiver scenarios.
  - We won’t dive into improving the group key exchange protocol for online communication like WhatsApp and MLS.
  - We don’t discuss post-quantum key exchange protocol improvement.
  - We don’t mention long-term keypair management or public key management on the message application server.
- Future Work
  - Do more evaluation on our scheme, and deploy it on smart devices in the future.

Thank You