The LASER Workshop

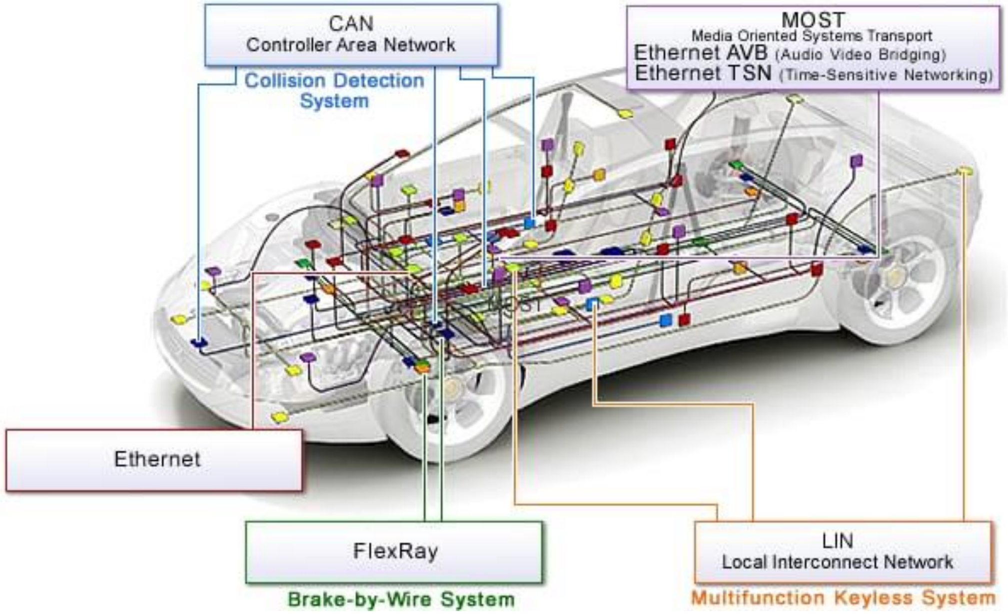
Session Key Distribution Made Practical for CAN and CAN-FD Message Authentication — Lessons Learned from Experiment

Yang Xiao, Shanghao Shi, Ning Zhang, Wenjing Lou, Y. Thomas Hou

Presenter: Yang Xiao <xiaoy@vt.edu>

Controller Area Network (CAN)

- Automotive Communication Networks
  - For in-vehicle communication between Electronic Control Units (ECUs)

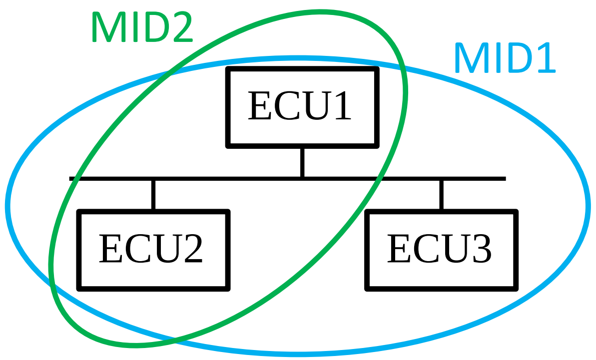


- Controller Area Network (CAN bus)
  - Used for the control of powertrain or other safety-critical subsystems
  - Extension: CAN Flexible Data-rate (CAN FD)

Image source: https://www.renesas.com/us/en/solutions/automotive/technology/networking-solutions.html

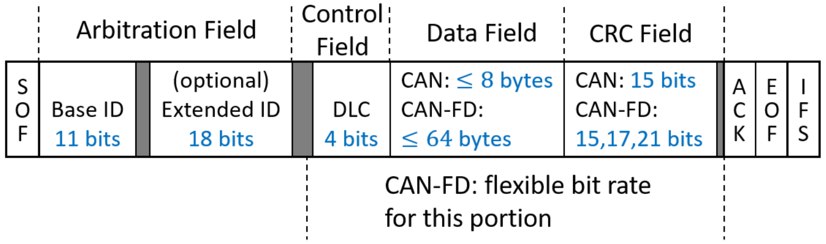
Basics of CAN and CAN FD Messaging

- Broadcast-and-Subscribe Messaging Paradigm
  - Only message ID, no sender or receiver ID

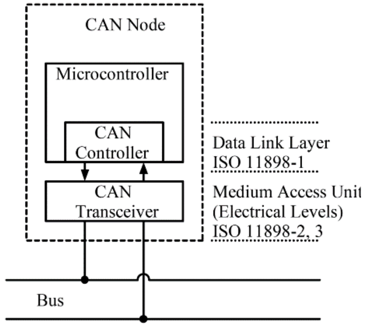


ECU-Message ID Subscription Example

- Physical Layer
  - CAN: fixed data rate
  - CAN FD: flexible data rate for data + CRC fields



CAN/CAN FD Data Frame Format

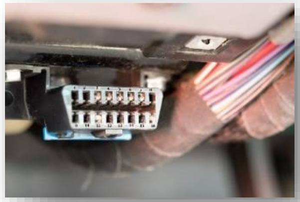


CAN/CAN FD Node Architecture

Image credit: https://en.wikipedia.org/wiki/CAN_bus

Attack on Vehicles

- Gain Access to Internal Control of Vehicle
  - Through the OBD-II port or exposed wired/wireless interfaces
  - → Eavesdrop
  - → Spoof messages to critical ECUs
  - → Knocking a ECU offline



OBD-II port

Hacker Finds He Can Remotely Kill Car Engines After Breaking Into GPS Tracking Apps

"I can absolutely make a big traffic problem all over the world," the hacker said.

ANDY GREENBERG SECURITY 08.01.2016 03:30 PM

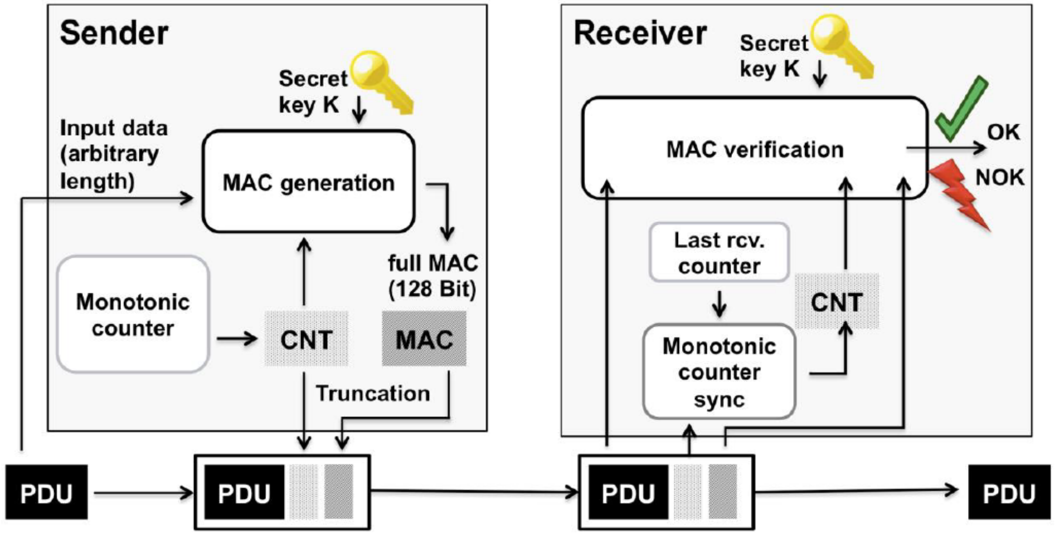
The Jeep Hackers Are Back to Prove Car Hacking Can Get Much Worse

After sparking a 1.4 million vehicle Chrysler recall, the security researchers offer a new lesson: It could have been---and could still be---much worse.

- Root Cause
  - No security mechanism built in the communication protocol
  - "Security by obscurity" is no longer safe for in-car systems

AUTOSAR Specifications on Secure Communication

- Security Goal Specified in AUTOSAR-SecOC
  - ECU entity authentication
  - Message authentication
    - Freshness – for replay attack resistance
    - Each message ID is assigned a MAC key
  - Cryptography (symmetric)
    - 128-bit keys
    - 64-bit MACs



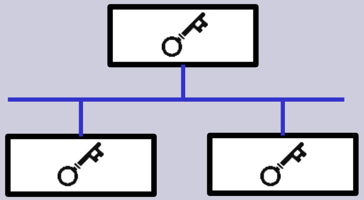
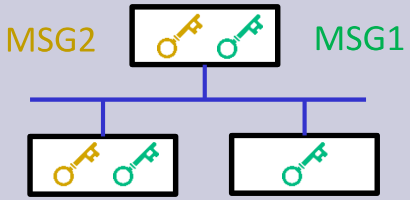
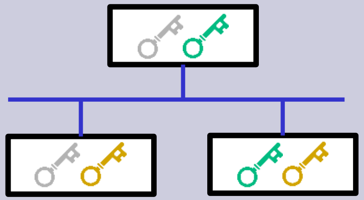
Message authentication with freshness verification [SecOC 4.2.2]

- What is Missing – Specification on Session Key Establishment for MAC Purposes
  - Critical to real-world deployment
  - → **Goal of this work**

[SecOC 4.2.2] AUTOSAR, "AUTOSAR release 4.2.2: Specification of module secure onboard communication," 2017

On Key Management and Establishment

AUTOSAR Compliant

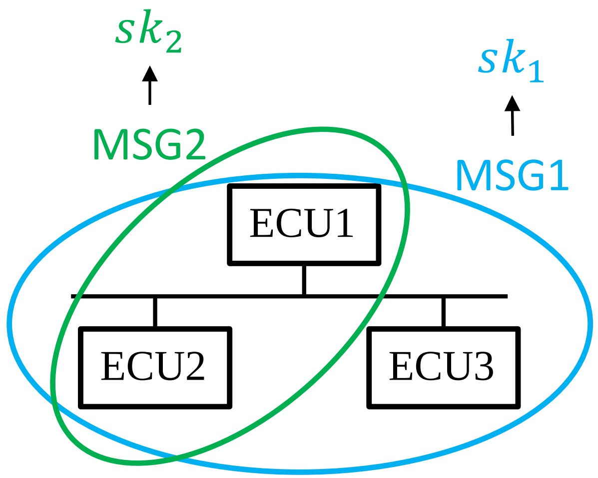
	One Key for All		One Key for Each MSG ID		ECU-pairwise Keys
					
Security Goal:	Auth. to send		Auth. to send a specific message		Entity auth.
Session Key Storage per ECU:	One		# of MSG subscribed		ECU population

Key Establishment Styles [Communication Complexity]

Key Agreement:	High	**(No LT key needed)**	High		High
Key Derivation:	Low	**(Msg-specific LT keys needed)**	Low		Low
Key Distribution:	Low	**(ECU-specific LT keys needed)**	Medium	**Our proposal fits here**	High

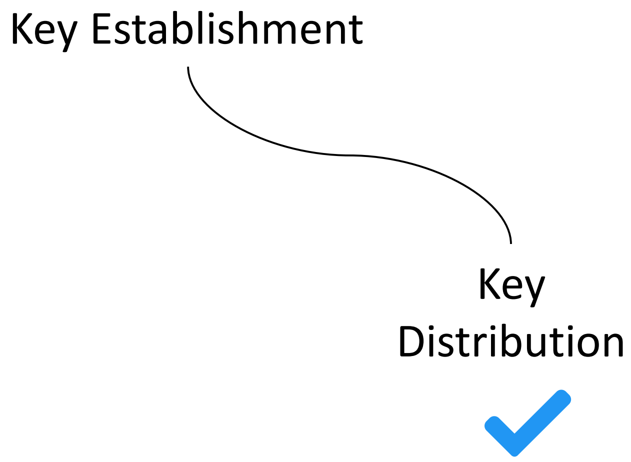
Our Proposed Key Management Architecture

- **System Model**
  - N ECUs, M message IDs
  - ECU i has a Subscription List ($\mathbf{SL}_i$) of message IDs
  - **Goal:** All ECUs subscribing message j shall get a shared session key sk_j



- **Threat Model**
  - Message eavesdropping, tampering, spoofing and replaying in the bus

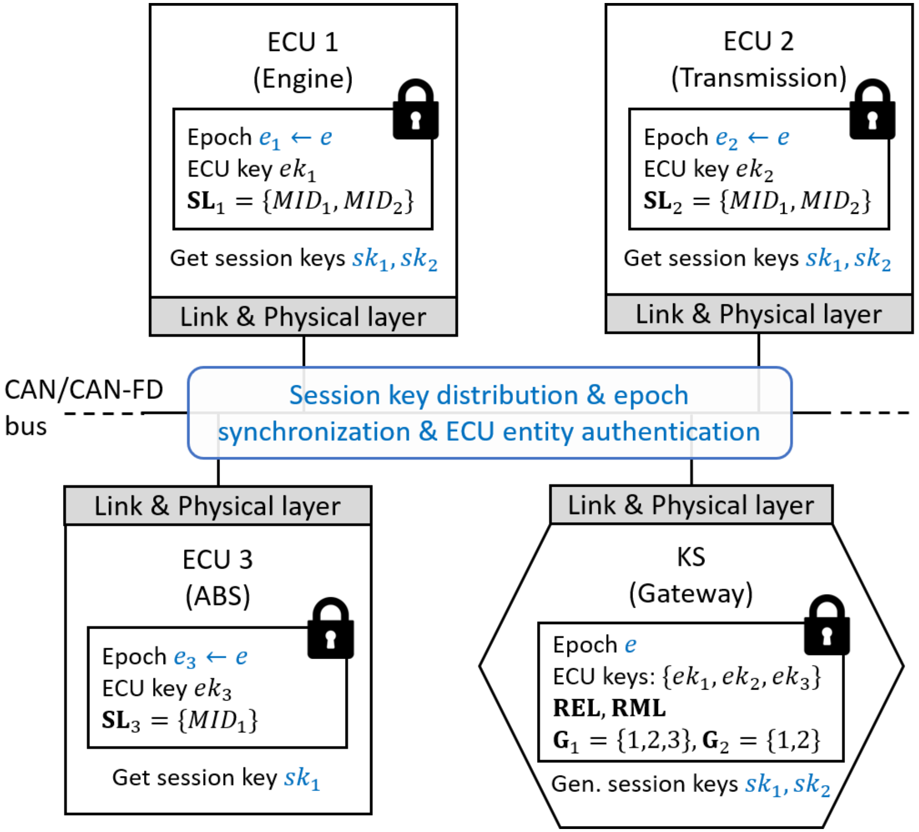
- **Practical Requirements for Key Establishment**
  - **R1:** Lightweight Computation and Storage
  - **R2:** Communication Efficiency
  - **R3:** AUTOSAR-compliant Security
  - **R4:** Flexibility with On-demand ECU



Our Proposed Key Management Architecture

- Key Server (KS)
  - Shares a long-term key ek_i with every ECU i
  - To generate 128-bit session keys $sk_1, \ldots, sk_M$
  - To maintain the 64-bit system epoch e

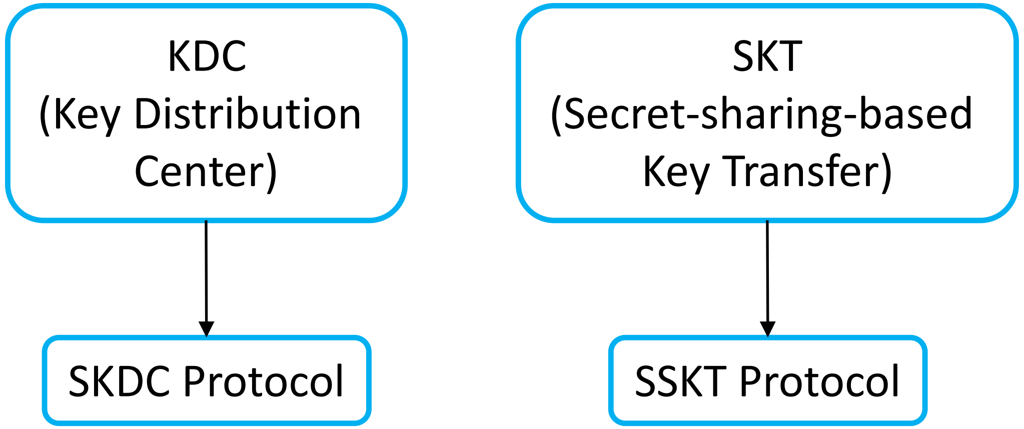
Example:
ECU 1, 2, 3 subscribe msg 1
ECU 1, 2 subscribes msg 2



- Key Distribution Protocols

Primitives:

Adapted for session key distribution in CAN/CAN-FD:



Baseline

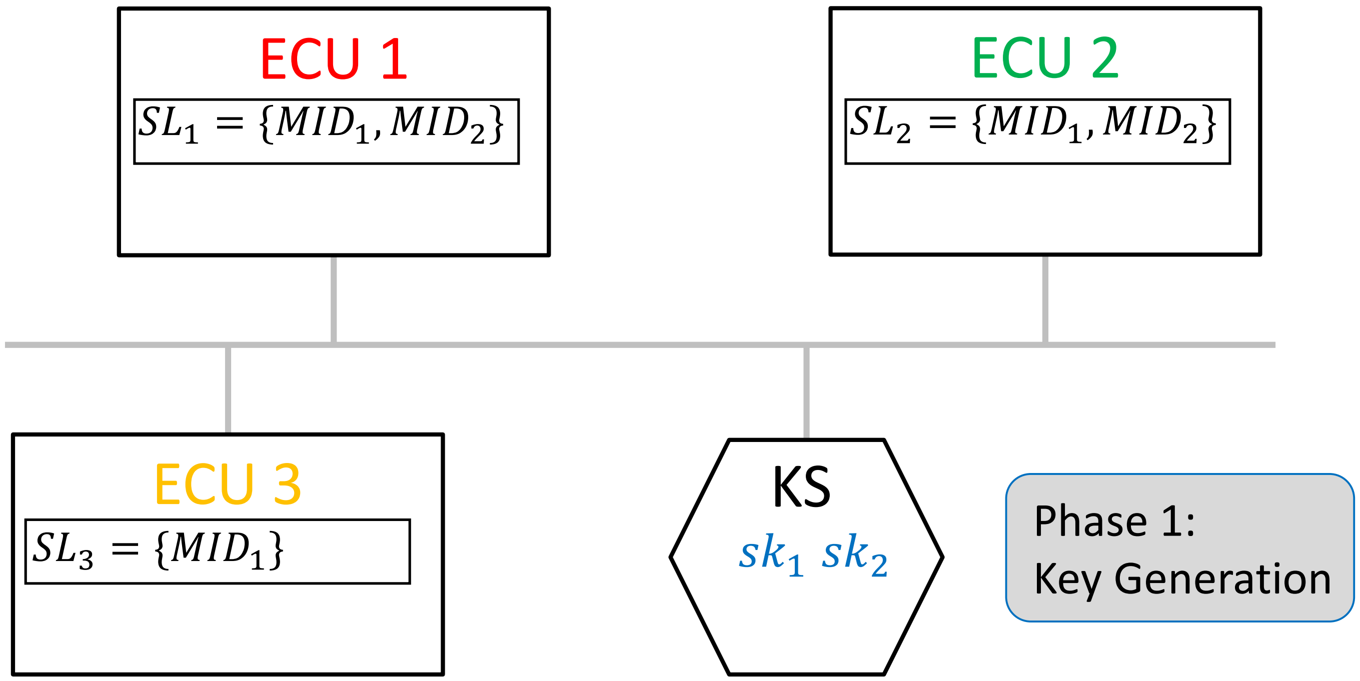
Novel Scheme
(for better communication efficiency)

Protocol Workflow, Experiment and Evaluation

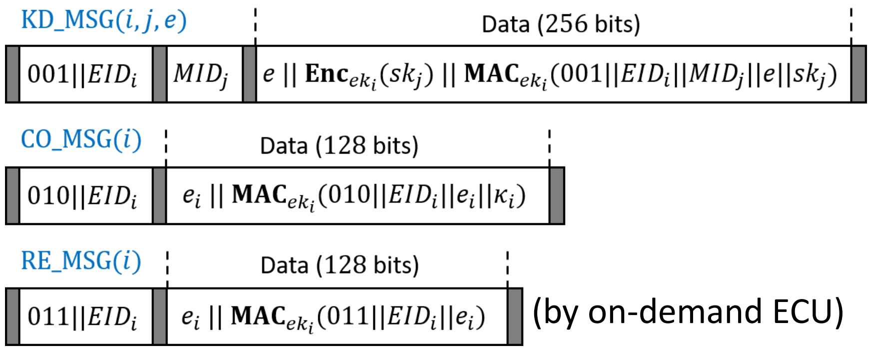
SKDC Protocol (baseline)

- Highlights
  - KS uses ek_i as key-encryption key (KEK) to encrypt each session key to each ECU i
  - Epoch e for freshness; MAC for verification

- Workflow (Example)

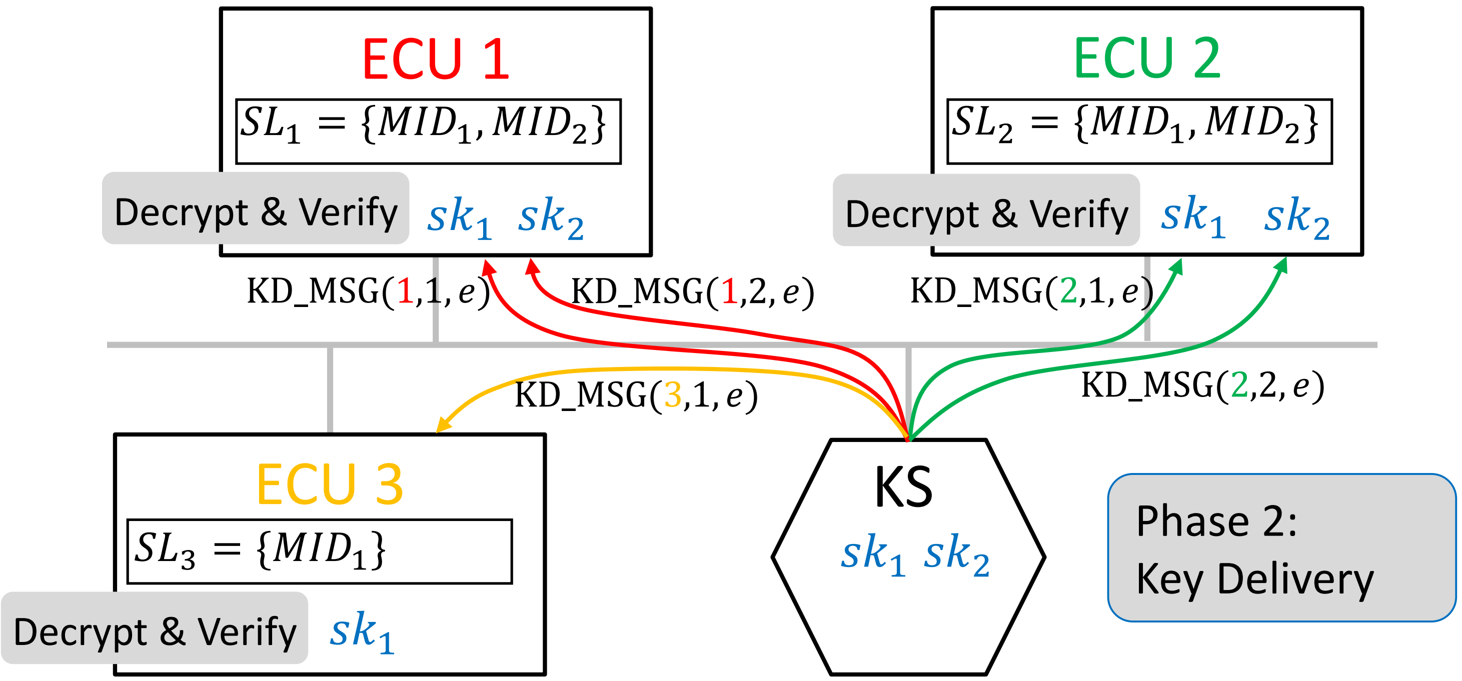


SKDC Protocol Message Formats

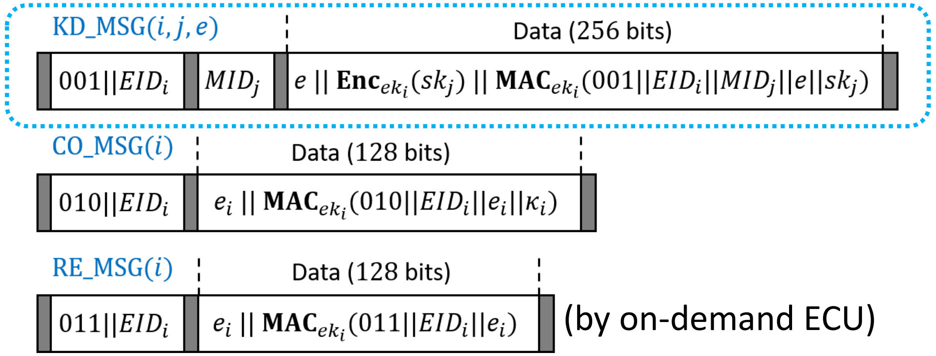


SKDC Protocol (baseline)

- Highlights
  - KS uses ek_i as key-encryption key (KEK) to encrypt each session key to each ECU i
  - Epoch e for freshness; MAC for verification
- Workflow (Example)



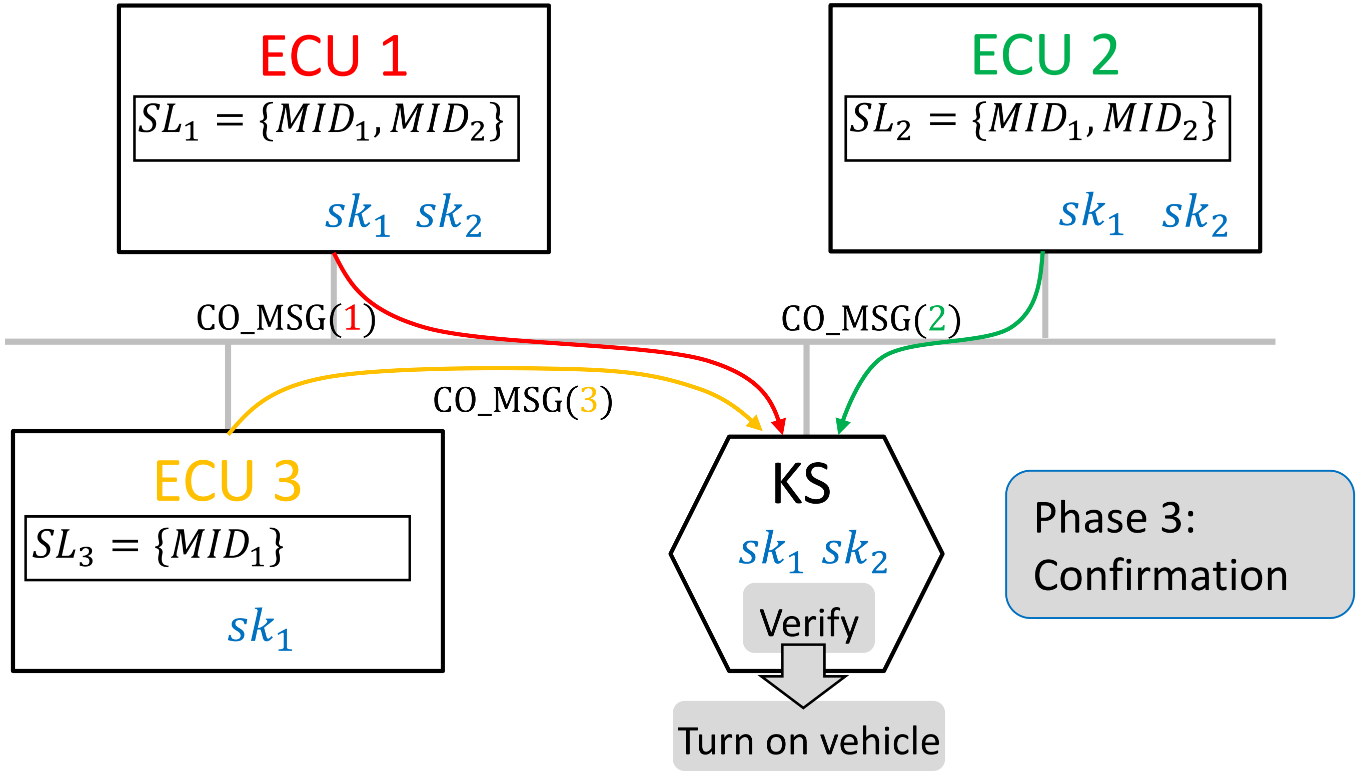
SKDC Protocol Message Formats



SKDC Protocol (baseline)

- Highlights
  - KS uses ek_i as key-encryption key (KEK) to encrypt each session key to each ECU i
  - Epoch e for freshness; MAC for verification

- Workflow (Example)



SKDC Protocol Message Formats

KD_MSG(i, j, e) — Data (256 bits)

$001\|\|EID_i$	MID_j	$e \,\|\|\, \mathbf{Enc}_{ek_i}(sk_j) \,\|\|\, \mathbf{MAC}_{ek_i}(001\|\|EID_i\|\|MID_j\|\|e\|\|sk_j)$

CO_MSG(i) — Data (128 bits)

$010\|\|EID_i$	$e_i \,\|\|\, \mathbf{MAC}_{ek_i}(010\|\|EID_i\|\|e_i\|\|\kappa_i)$

RE_MSG(i) — Data (128 bits)

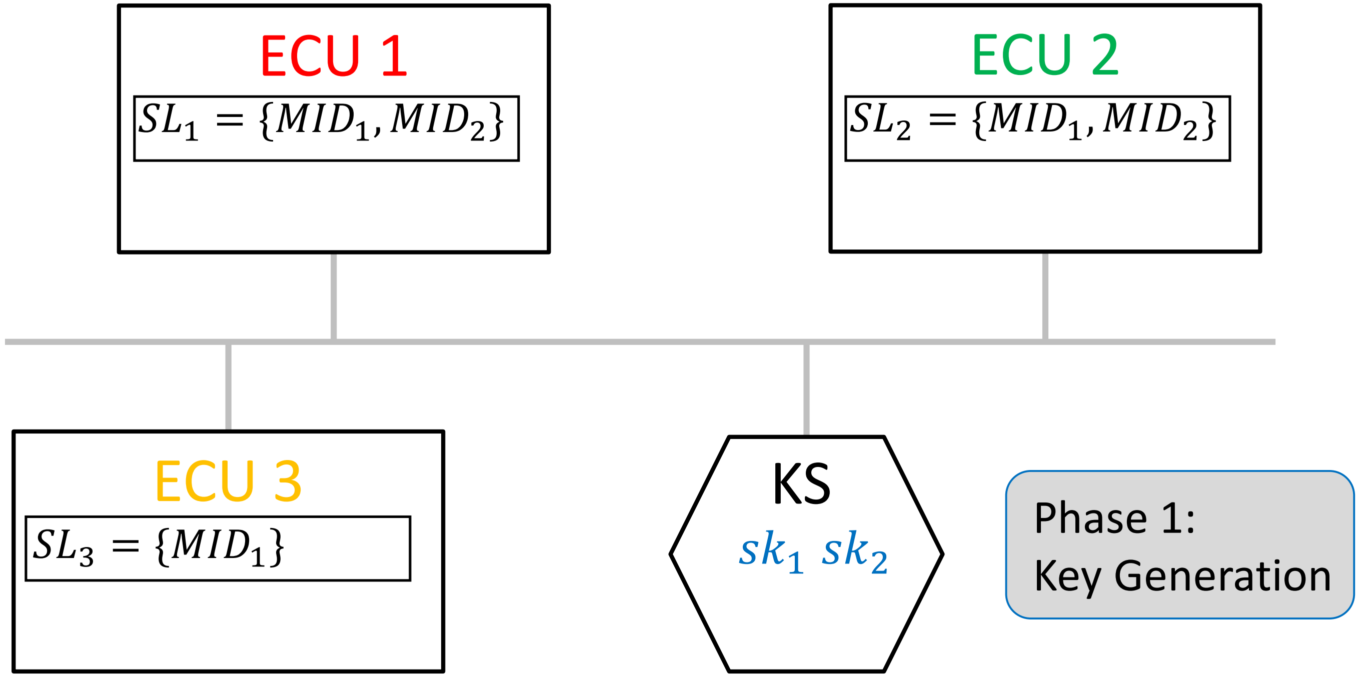
$011\|\|EID_i$	$e_i \,\|\|\, \mathbf{MAC}_{ek_i}(011\|\|EID_i\|\|e_i)$

(by on-demand ECU)

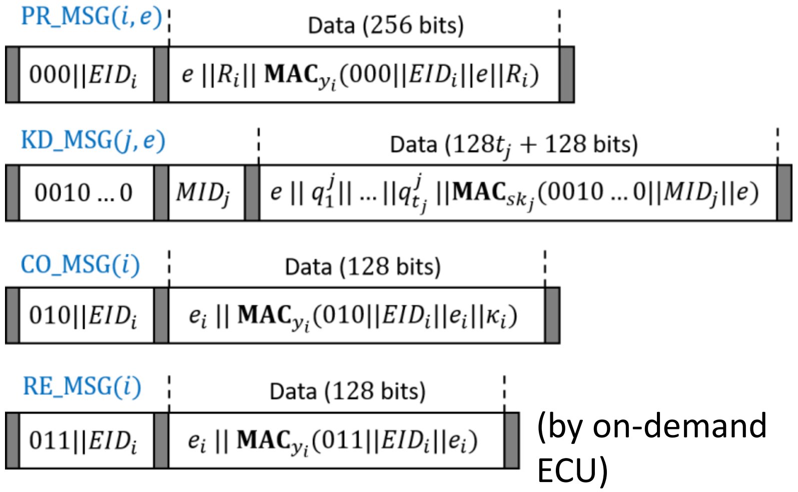
- Protocol message is sent in separate CAN/CAN-FD frames if payload exceeds frame limit.
- On-demand ECU sends RE_MSG during driving for requesting session keys.

SSKT Workflow

- Highlights
  - Long-term key pair $ek_i = (x_i, y_i)$
  - ECU recovers session key segments by Lagrange polynomial interpolation in $GF(256)$
- Workflow (Example)

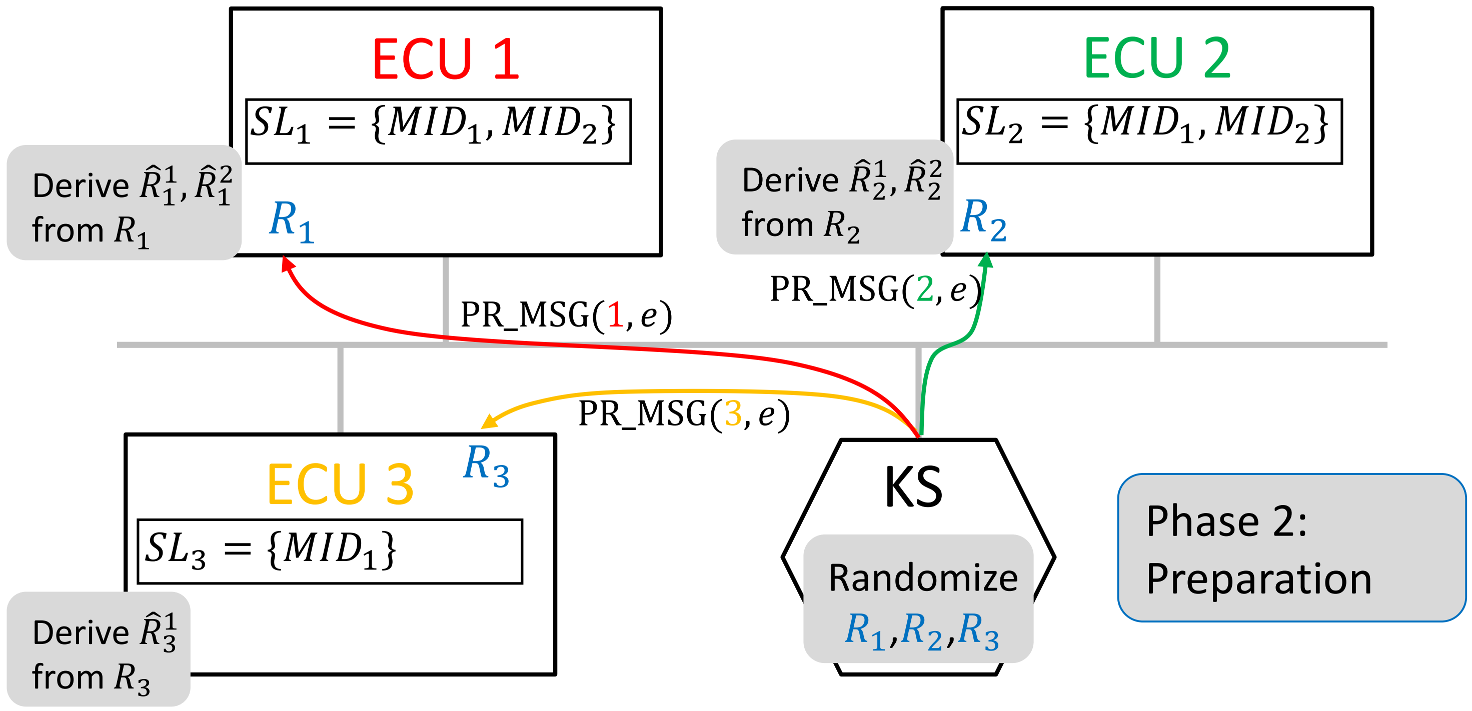


SKDC Protocol Message Formats

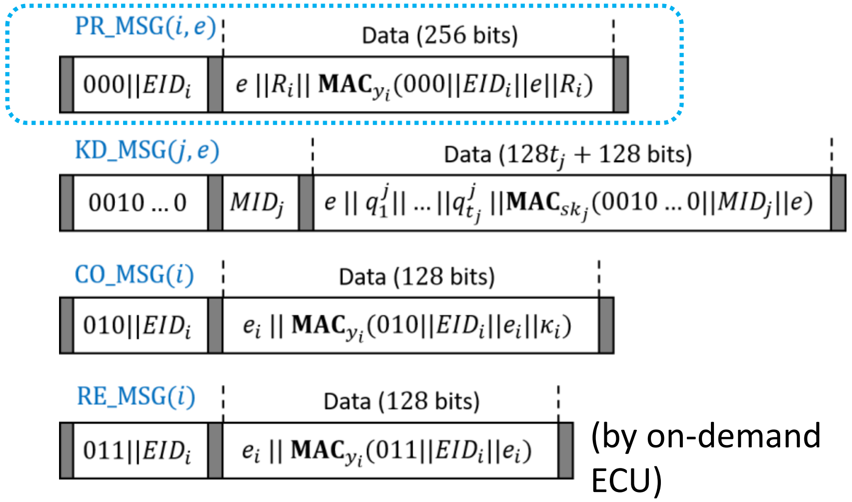


SSKT Workflow

- Highlights
  - Long-term key pair $ek_i = (x_i, y_i)$
  - ECU recovers session key segments by Lagrange polynomial interpolation in $GF(256)$
- Workflow (Example)

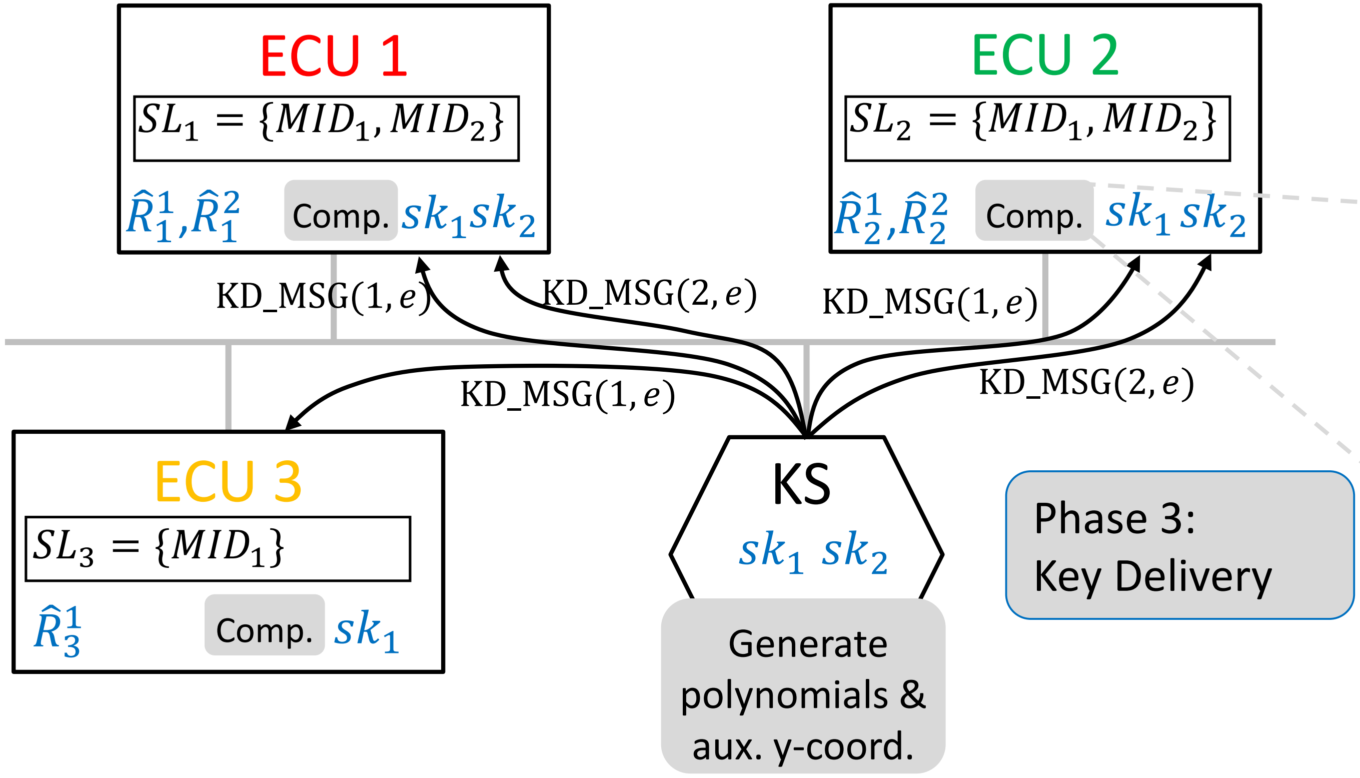


SKDC Protocol Message Formats

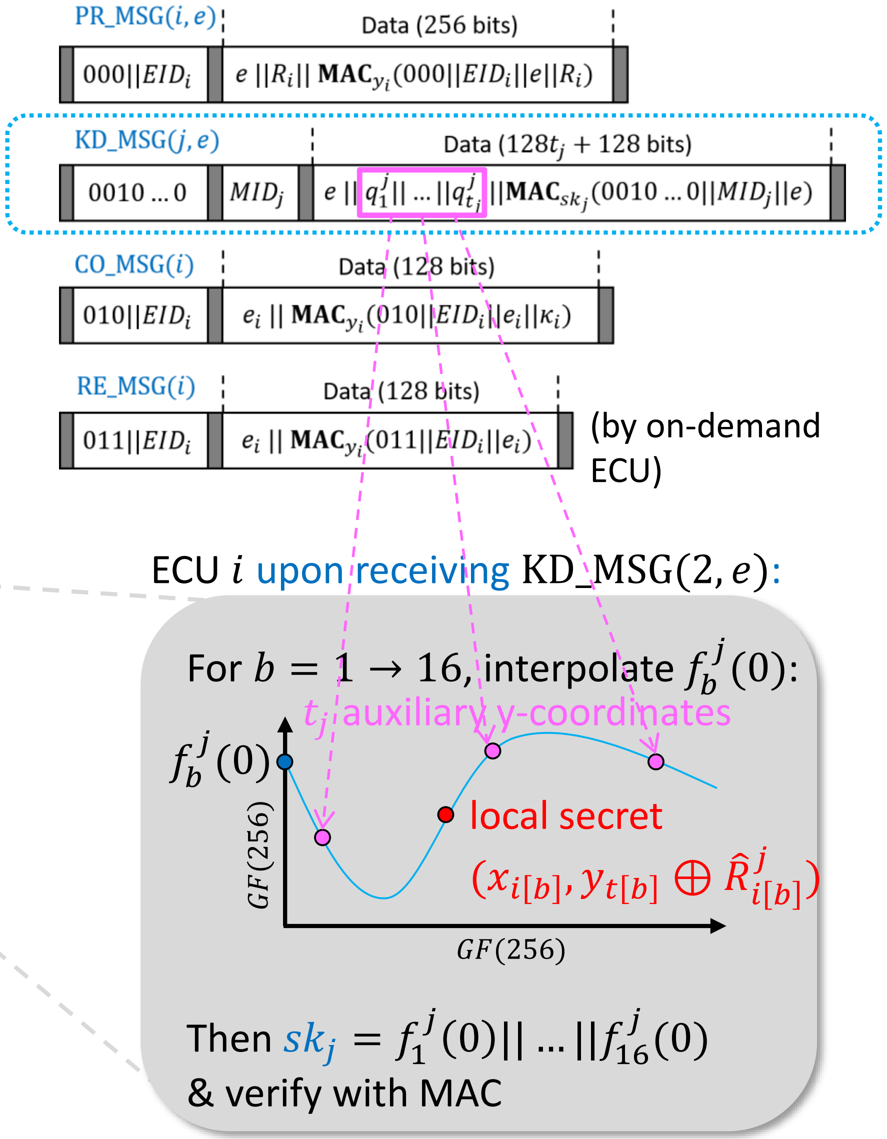


SSKT Workflow

- Highlights
  - Long-term key pair $ek_i = (x_i, y_i)$
  - ECU recovers session key segments by Lagrange polynomial interpolation in $GF(256)$
- Workflow (Example)

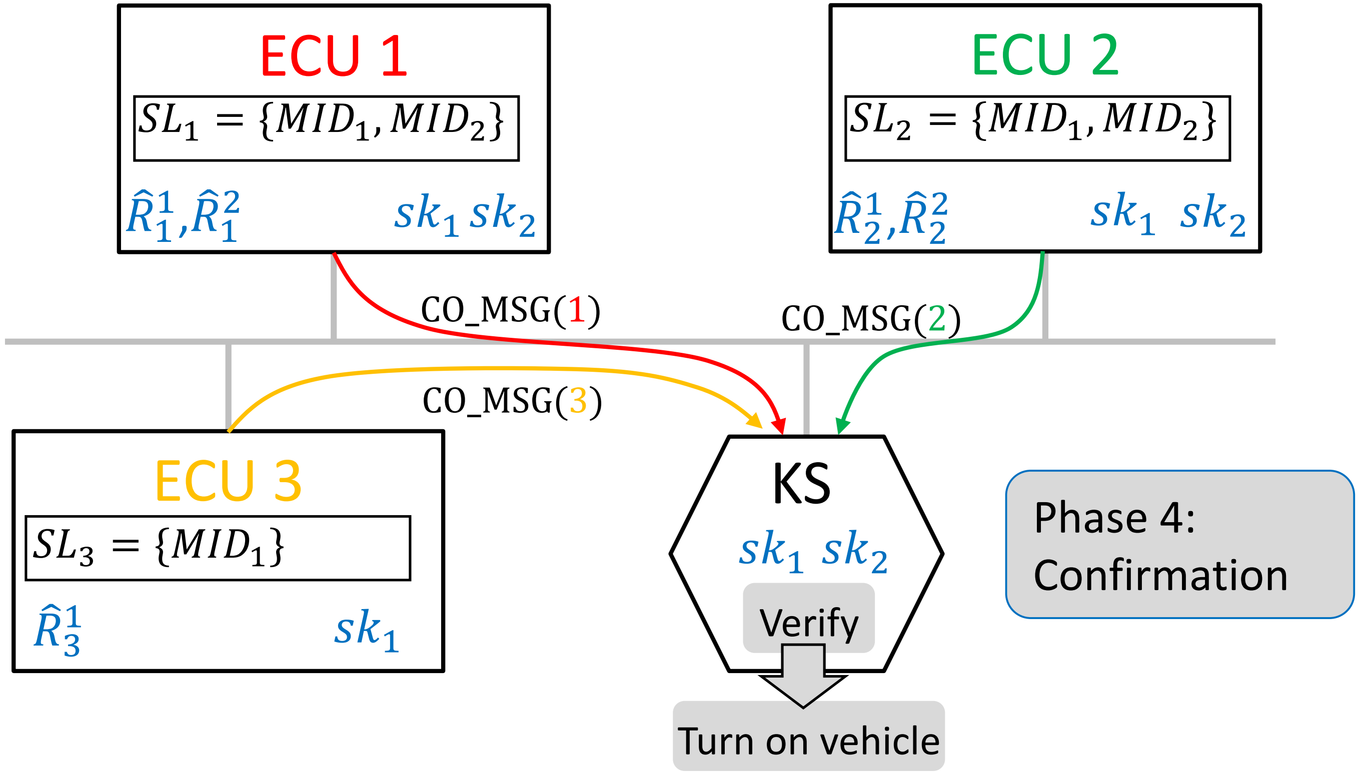


SKDC Protocol Message Formats

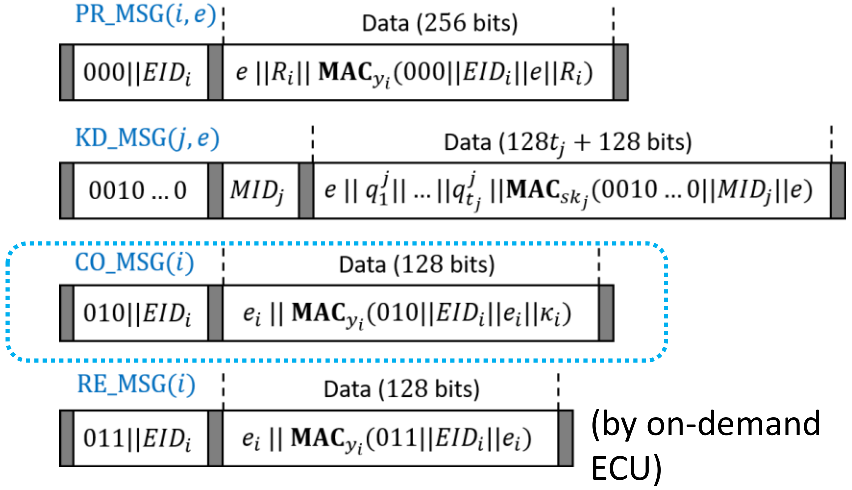


SSKT Workflow

- Highlights
  - Long-term key pair $ek_i = (x_i, y_i)$
  - ECU recovers session key segments by Lagrange polynomial interpolation in $GF(256)$
- Workflow (Example)



SKDC Protocol Message Formats

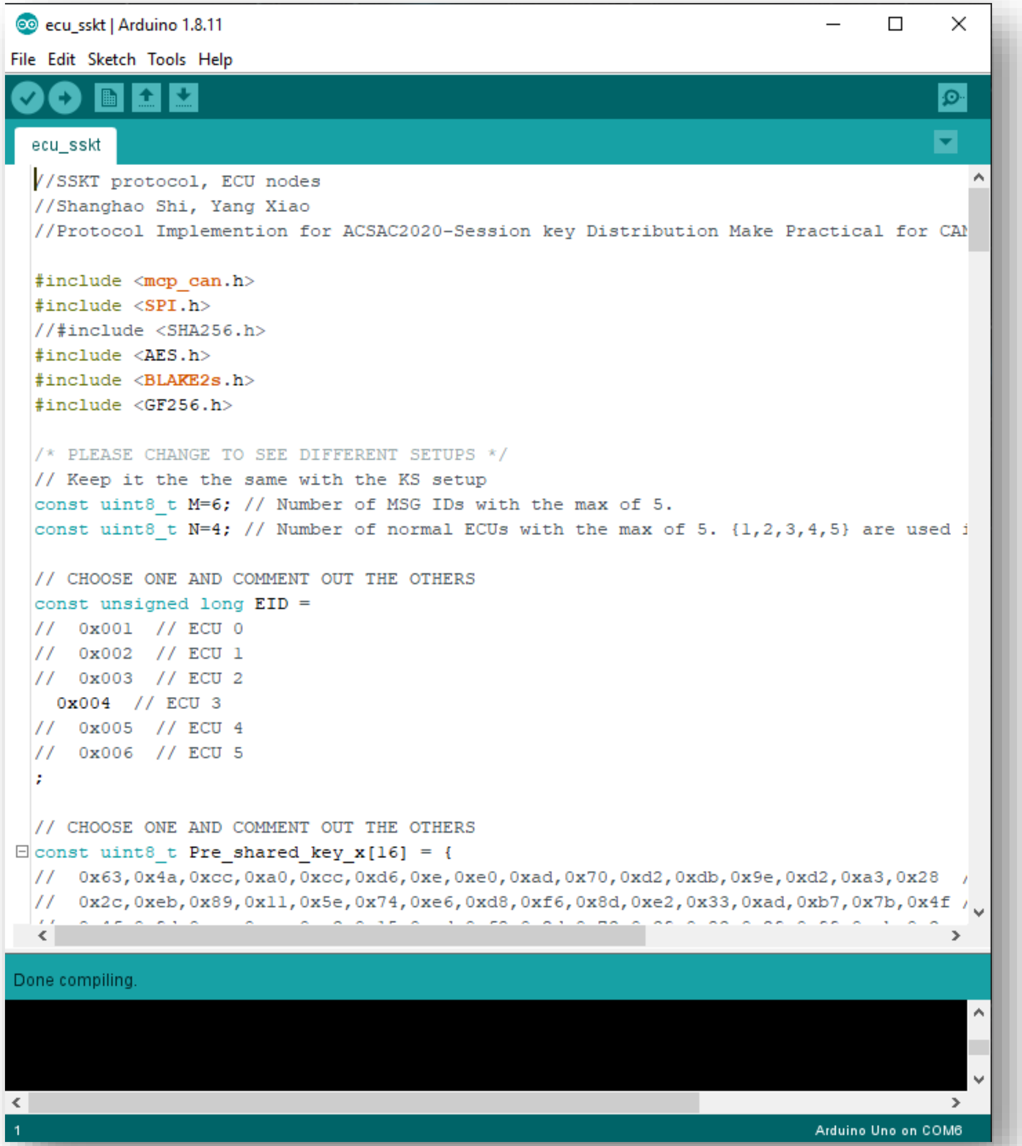


- Protocol message is sent in separate CAN/CAN-FD frames if payload exceeds frame limit.
- On-demand ECU sends RE_MSG during driving for requesting session keys (same as SKDC).

Implementation for CAN Bus Deployment

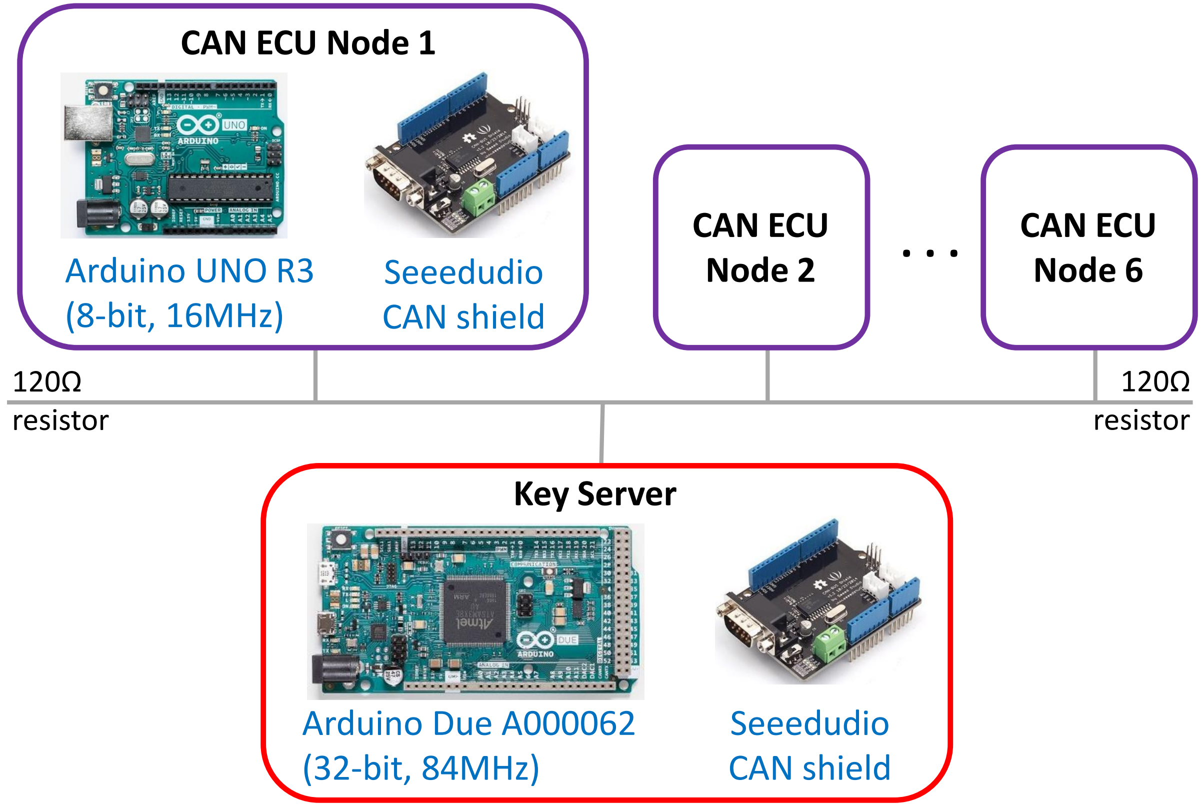
- *Keyserver* and *node* Programs
  - Arduino IDE (C++)
- Third-party libraries used
  - Arduino Cryptography Library (rweather.github.io/arduinolibs/crypto.html)
  - Seeed Studio CAN Bus Shield (github.com/Seeed-Studio/CAN_BUS_Shield)

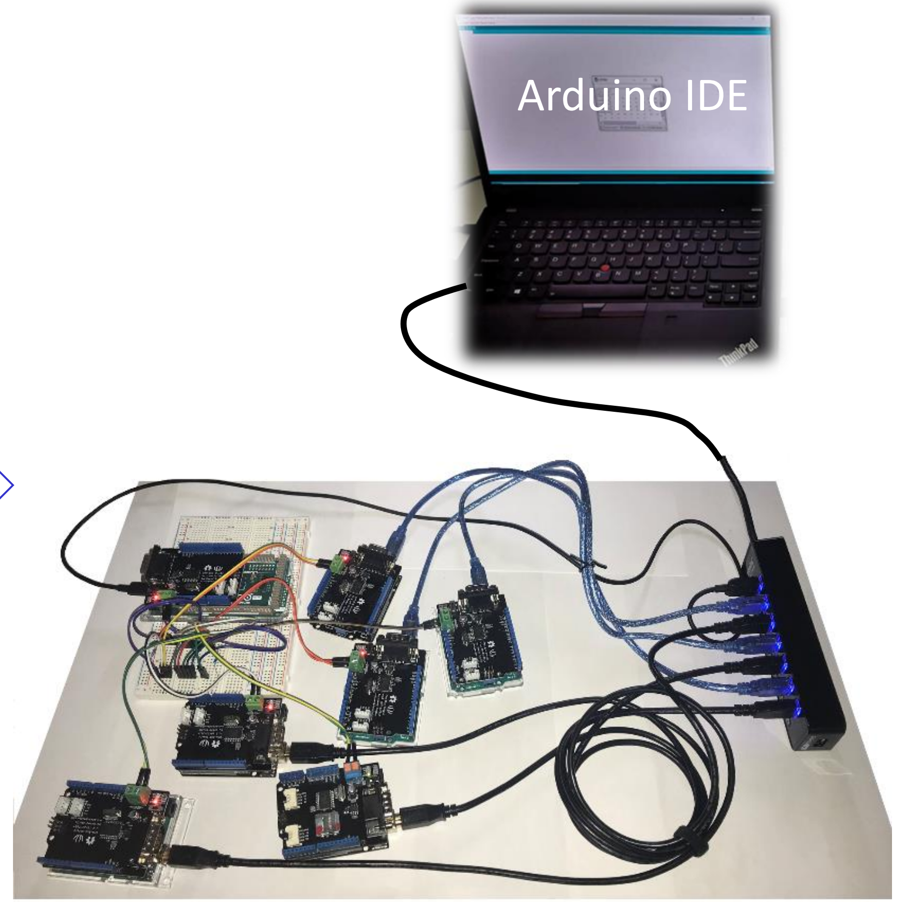
Code: github.com/yang-sec/CAN-SessionKey



Test Platform with CAN Bus

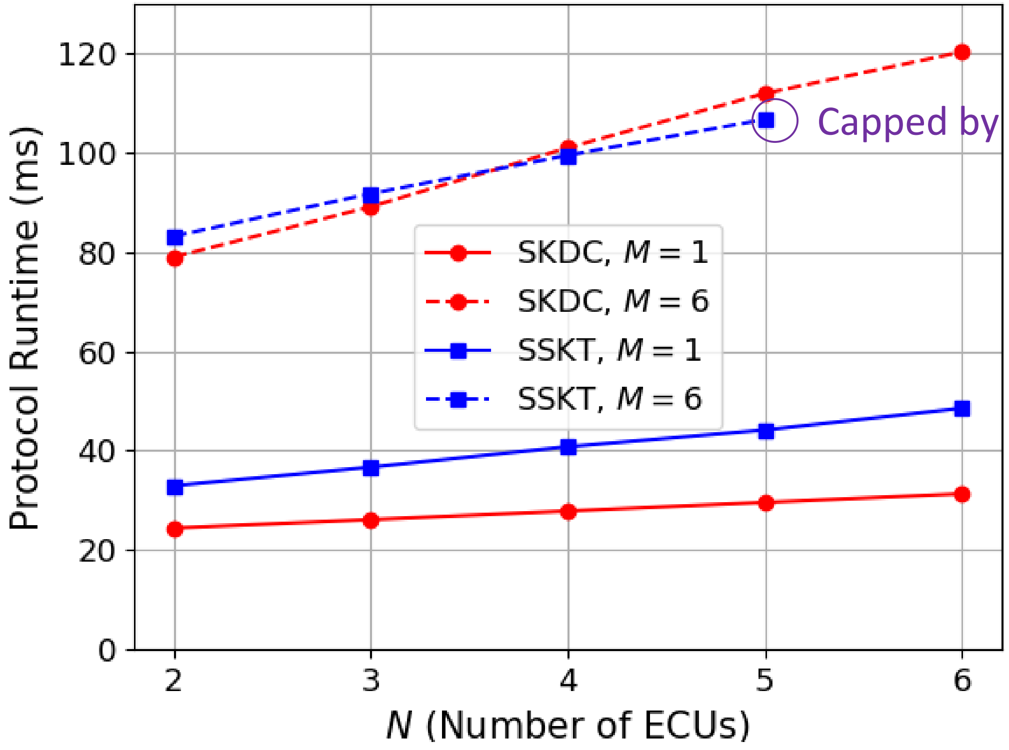
- Setup
  - $N \in \{1,2,3,4,5,6\}$ ECUs, each subscribes to all $M \in \{1,6\}$ MIDs
  - Data collected using serial terminals of Arduino IDE





Hardware Experiment Result

- Runtime Results for Distributed One Session Key (ms)



Capped by N=5 due to Uno's RAM limit

SSKT's advantage will continue scaling up for larger M and N, contributed by its better communication efficiency in the key delivery phase.

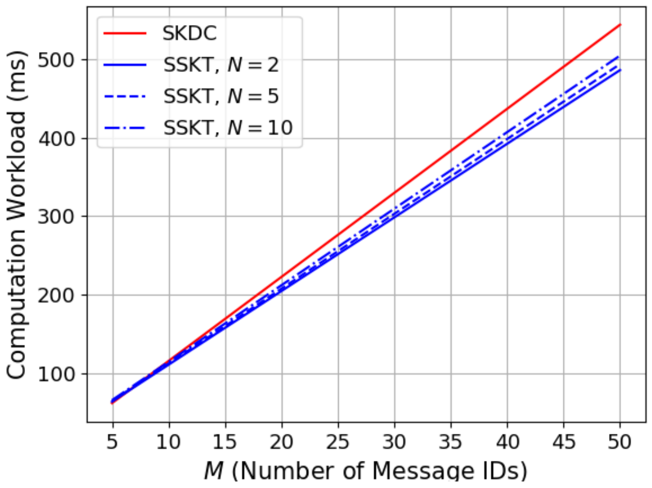
Performance Extrapolation – Computation Workload

- Single-operation Runtimes
  - Evaluated on normal ECU (not KS)
  - Used both Uno and Due for benchmarking

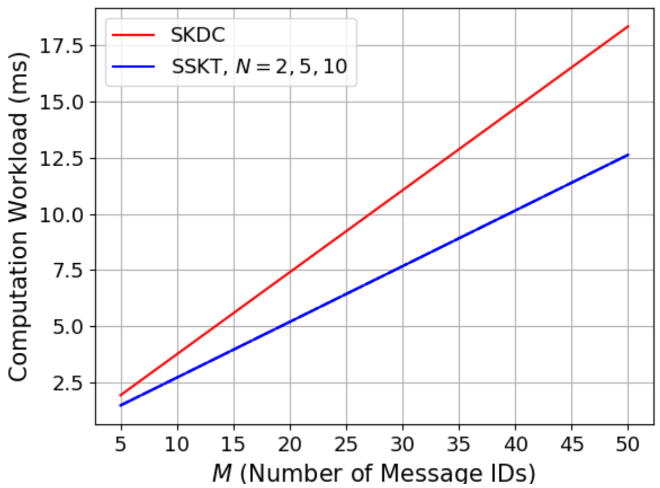
Operation	Uno	Due
AESSmall128-ECB Set Key	131.64	23.66
AESSmall128-ECB Encrypt (per byte)	42.40	7.06
AESSmall128-ECB Decrypt (per byte)	73.66	12.33
AESTiny128-ECB Set Key	9.98	1.25
AESTiny128-ECB Encrypt (per byte)	42.39	7.23
BLAKE2s Keyed Reset	3512.94	55.09
BLAKE2s Hash (per byte)	54.61	0.80
BLAKE2s Finalize	3508.25	53.14
Degree-2 Polynomial $f(0)$ Recovery (per byte)	10.40	~ 0
Degree-5 Polynomial $f(0)$ Recovery (per byte)	19.86	~ 0
Degree-10 Polynomial $f(0)$ Recovery (per byte)	33.56	~ 0

- Extrapolated ECU Computation Workload per Protocol Session

Arduino Uno as ECU



Arduino Due as ECU



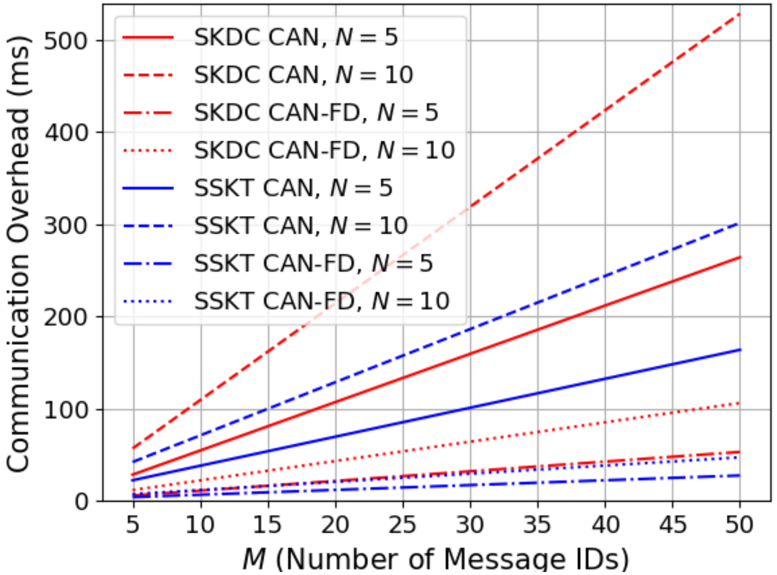
SSKT achieves better computation efficiency for larger M.
Tradeoff: RAM cost.

Performance Extrapolation – Communication Overhead

- Protocol Message Count per Protocol Session
  - Assume CAN-FD bit rate is 5 times of CAN's

		Message size (In CAN bits)	Message count
SKDC (CAN):	KD_MSG	524	MN
	CO_MSG	222	N
SKDC (CAN-FD):	KD_MSG	105	MN
	CO_MSG	60	N
SSKT (CAN):	PR_MSG	444	N
	KD_MSG	$262(1+N)$	M
	CO_MSG	222	N
SSKT (CAN-FD):	PR_MSG	86	N
	KD_MSG	avg. $60+39N$	M
	CO_MSG	60	N

- Extrapolated Communication Overhead per Protocol Session
  - Assume CAN bit rate is 500Kbs

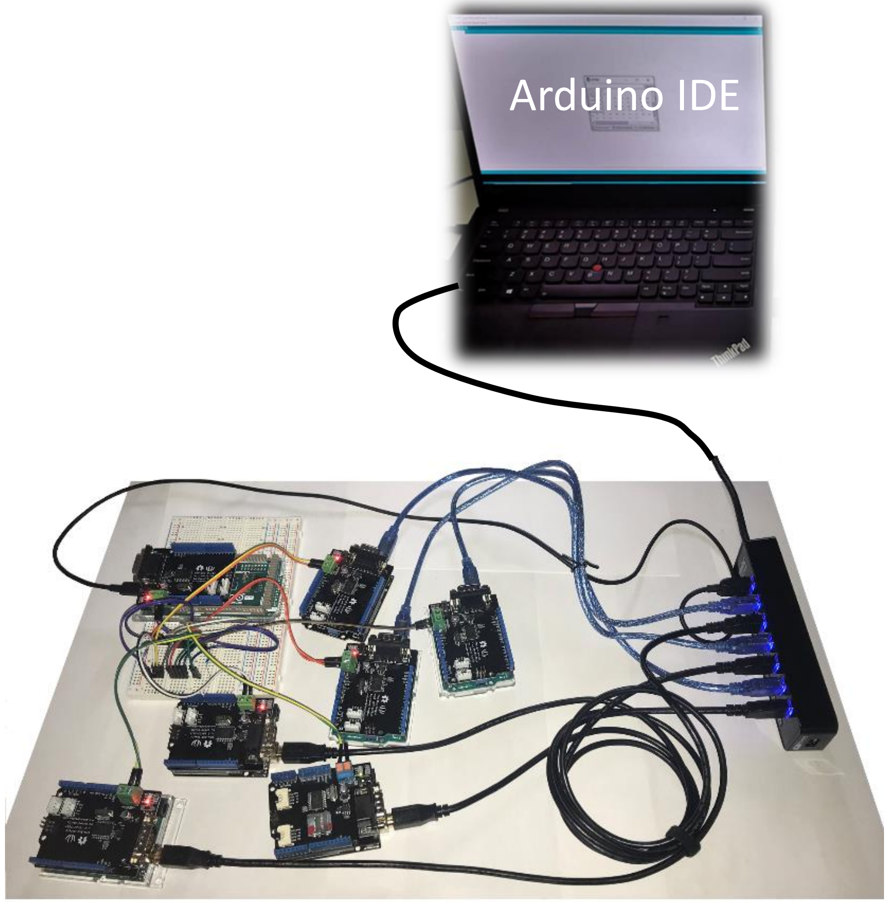


SSKT achieves better communication efficiency.

Discussions & Meta Questions

Experimentation Methodology

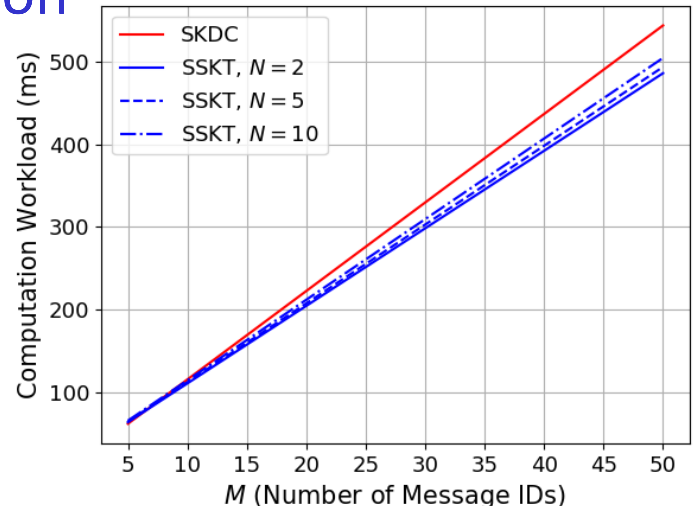
- Hardware Testbed Experiment



- Benchmarking

Operation	Uno	Due
AESSmall128-ECB Set Key	131.64	23.66
AESSmall128-ECB Encrypt (per byte)	42.40	7.06
AESSmall128-ECB Decrypt (per byte)	73.66	12.33
AESTiny128-ECB Set Key	9.98	1.25
AESTiny128-ECB Encrypt (per byte)	42.39	7.23
BLAKE2s Keyed Reset	3512.94	55.09
BLAKE2s Hash (per byte)	54.61	0.80
BLAKE2s Finalize	3508.25	53.14
Degree-2 Polynomial $f(0)$ Recovery (per byte)	10.40	~ 0
Degree-5 Polynomial $f(0)$ Recovery (per byte)	19.86	~ 0
Degree-10 Polynomial $f(0)$ Recovery (per byte)	33.56	~ 0

- Extrapolation Analysis



Experimentation Artifacts Usage

- Standard Cryptography
  - **Arduino Cryptography Library** (rweather.github.io/arduinolibs/crypto.html)
  - For lightweight embedded systems

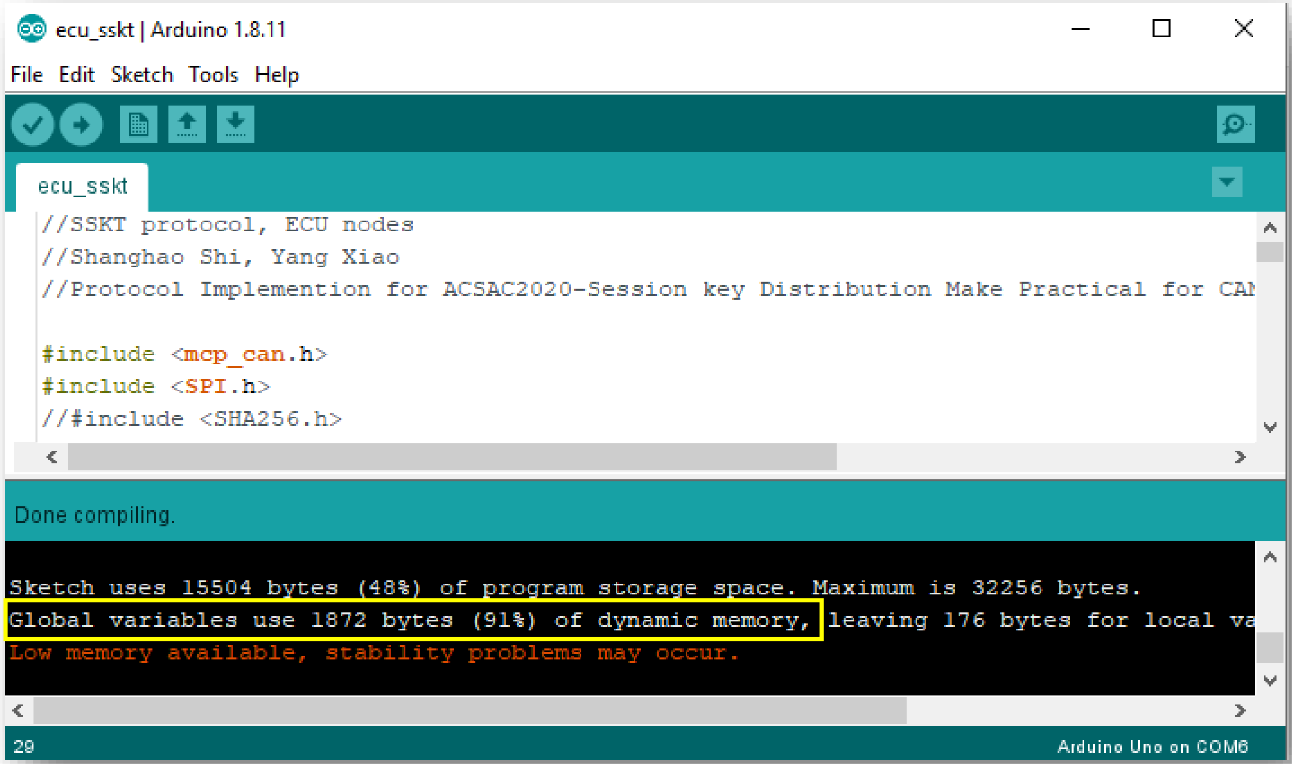
Arduino UNO R3
(8-bit, 16MHz)

- CAN Bus Functionalities
  - **Seeed Studio CAN Bus Shield** (github.com/Seeed-Studio/CAN_BUS_Shield)

Seeed Studio
CAN shield

Setbacks and Challenges Encountered in HW Experiment

- Unstable Arduino board performance
  - SRAM limits: 2MB in Arduino Uno



- Loss of CAN messages
  - When multiple messages are received – caused by limited buffer size
  - An intrisic CAN messaging problem
  - **Solution:** tweaking protocol message timing (tricky business)

Tradeoff in Implementation (Standard Crypto)

- Benchmarks from Arduino Cryptography Library, on Arduino Uno

Encryption Algorithm	Encryption (per byte)	Decryption (per byte)	Key Setup	State Size (bytes)
AES128	33.28us	63.18us	158.68us	181
AESSmall128	40.37us	71.36us	134.22us	34
AESTiny128	40.37us		10.16us	18

Hash Algorithm	Hashing (per byte)	Finalization	Key Setup	State Size (bytes)
SHA256 (HMAC)	43.85us	8552.61us	2836.49us	107
SHA3-256	60.69us	8180.24us		205
BLAKE2s (keyed)	20.65us	1335.25us	1339.51us	107

Tradeoff in Implementation (Finite Field Arithmetic)

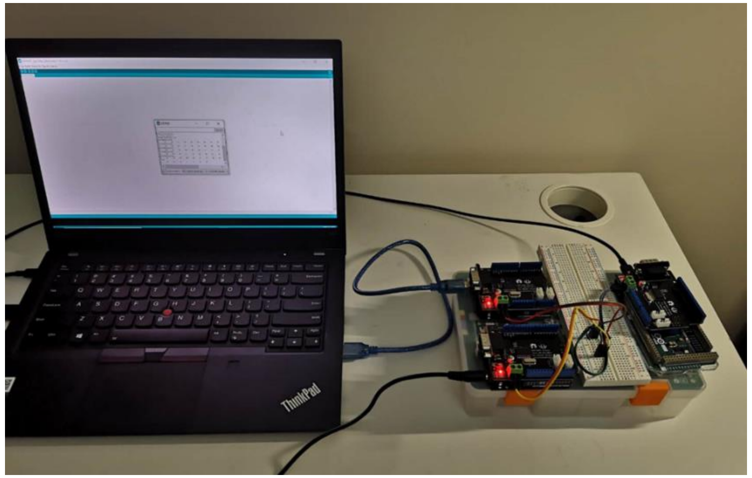
- Optimization for SSKT
  - Can we speed up polynomial interpolation?
  - → Yes but at a cost – trade space for time

- Three 16×16 lookup tables for $GF(256)$ arithmetic (784 bytes)
  - **Inverse table**
  - **Exponentiation table** and **logarithm table** (for realizing multiplication)

- Pre-computing Lagrange coefficients (16N bytes)
  - $f_b^j(0) = \sum_{m=1}^{t_j+1} v_m \left(\prod_{n=1,n\neq m}^{t_j+1} \frac{u_n}{u_n - u_m} \right)$

Previous Unanticipated Results

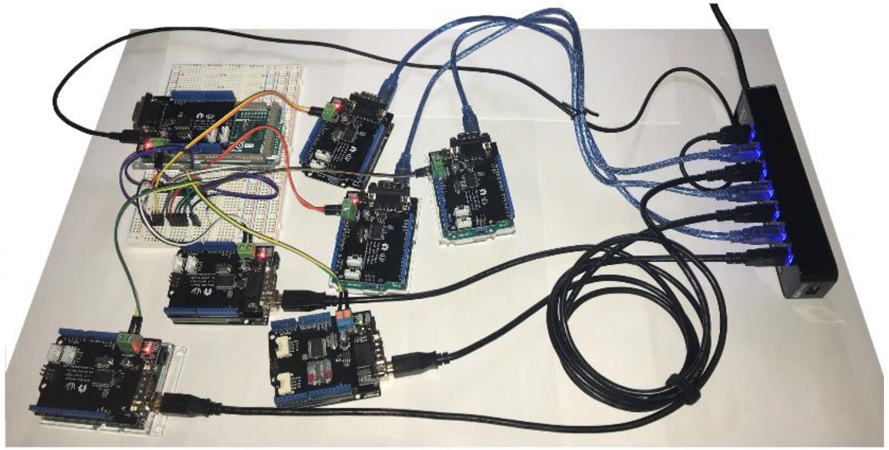
- Previous attempt – using one Arduino board to simulate multiple ECUs
  - Leaded to erroneous result!



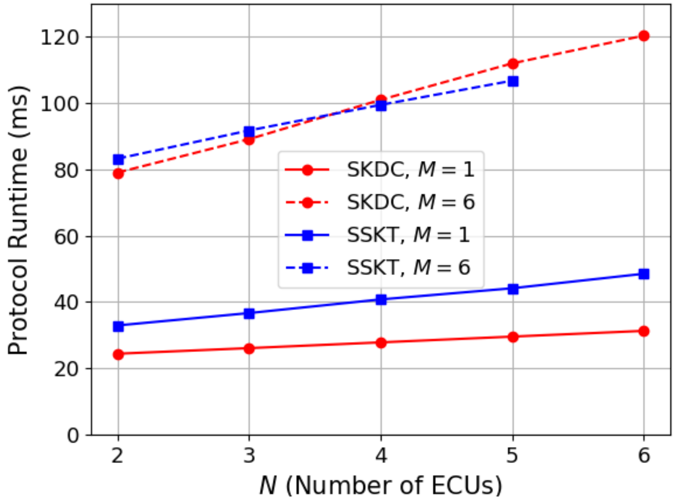
Protocol runtime (ms) – previous result

	$N = 2$	$N = 3$	$N = 4$	$N = 5$	$N = 6$
SKDC	9.021	12.708	17.659	21.998	25.855
SSKT	7.411	7.855	8.382	8.75	9.36

VS

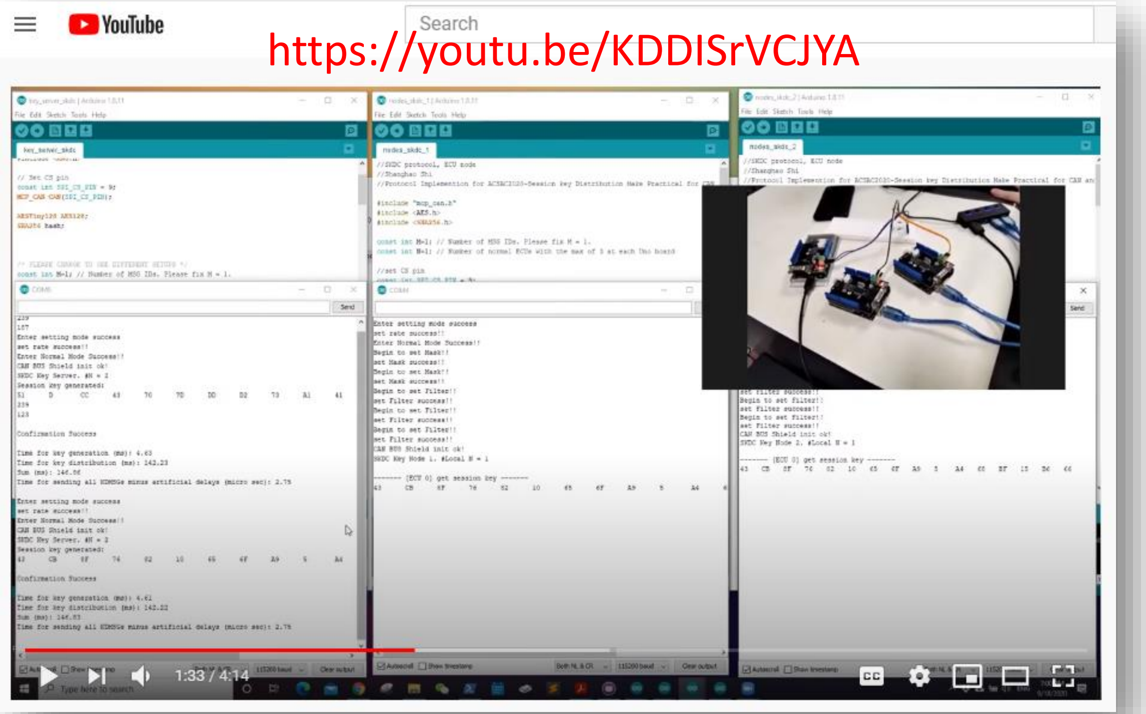


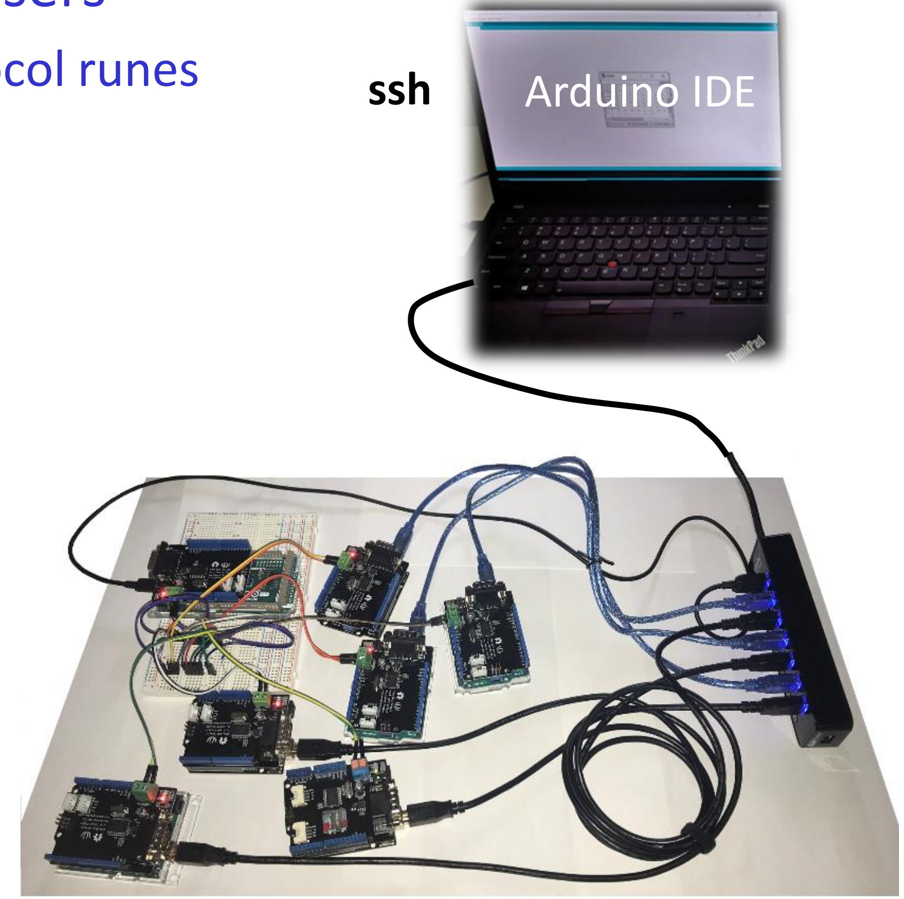
Protocol runtime (ms) – current result



Artifact Evaluation

- Making embedded system accessible for remote users
  - Good for benchmark evaluations and some simple protocol runes
- Still need human intervention in some cases!
  - So we did a live demonstration…





Wrap-up Discussion

Lessons Learned

- Hardware limitation → extrapolation from benchmark results
- Simulating a hardware environment is full of caveats
- Lightweight cryptography matters for cost-efficient embedded systems
- Overhead is significant when incepting security mechanisms in a legacy unsecure system (eg., CAN bus)

Future Directions (research + implementation)

- On Performance Bottleneck and Room for Improvement
  - Compared to computation workload, communication overhead has limited room for improvement
  - May use other automotive comm. network for evaluation
- On Storage and Memory Cost
  - SSKT achieves superior computation efficiency using pre-computed intermediate results, which needs SRAM to store
  - Though SRAM is affordable nowadays, the tradeoff deserves more attention
- On System Scalability
  - $GF(256)$-arithmetic caps the network size by 128
  - To support larger network size, need larger finite fields, eg., $GF(2^{16})$
  - Need more powerful ECUs
  - Need more efficient implementation of finite field arithmetic & polynomial operation
- Evaluation in Realistic Automotive Environment

Thanks!