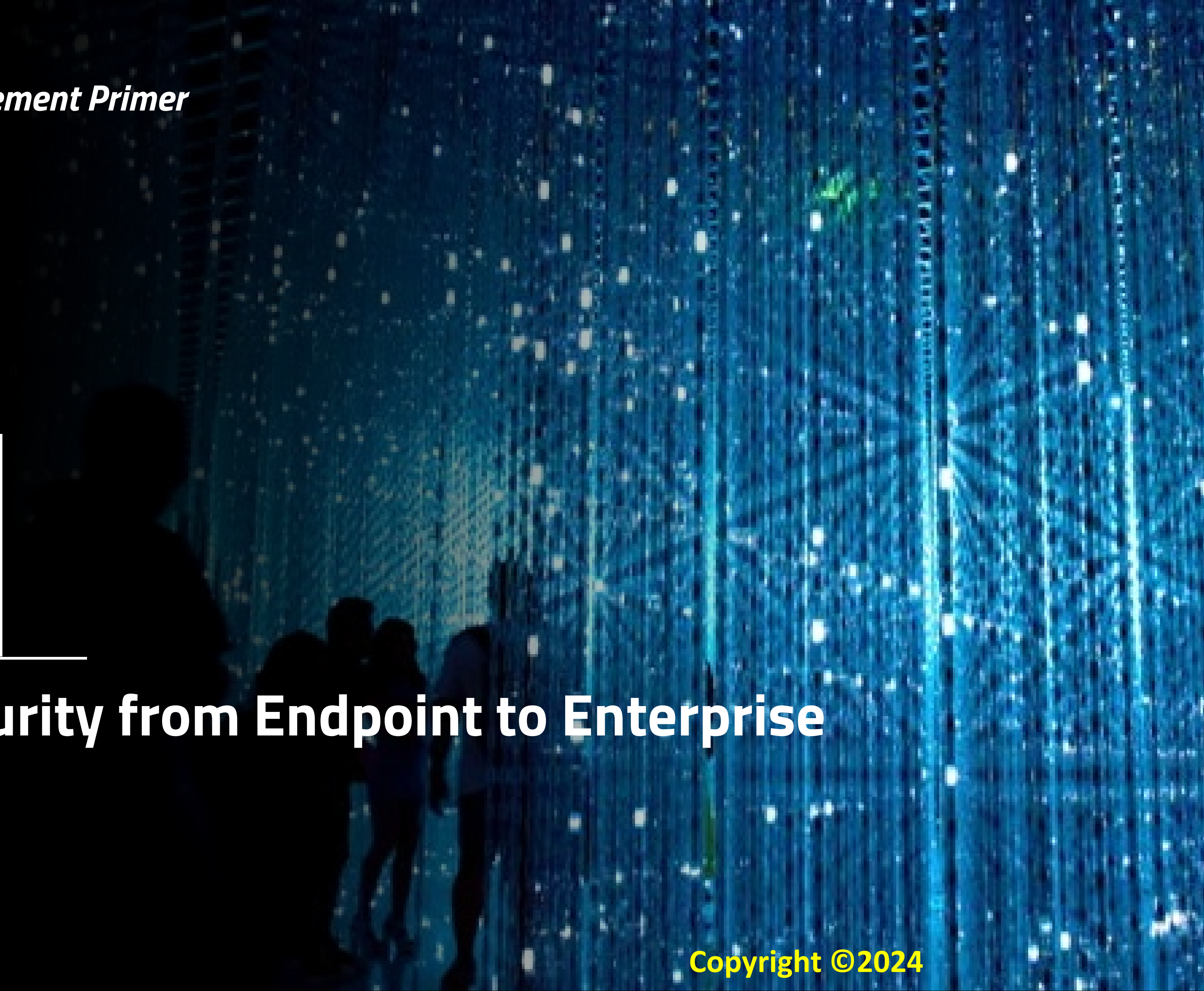
*Autonomous Key Management Primer*



AKM CYBER CORP

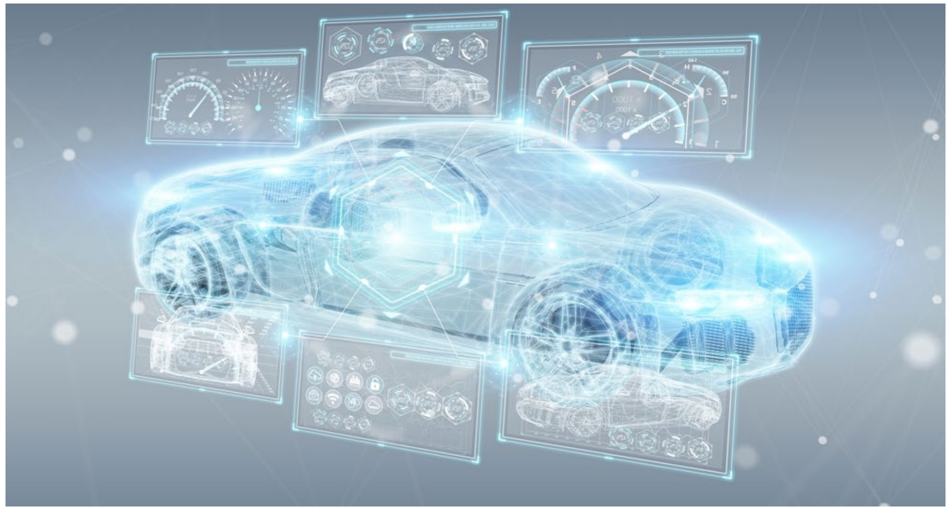
AUTOMATING CYBERSECURITY

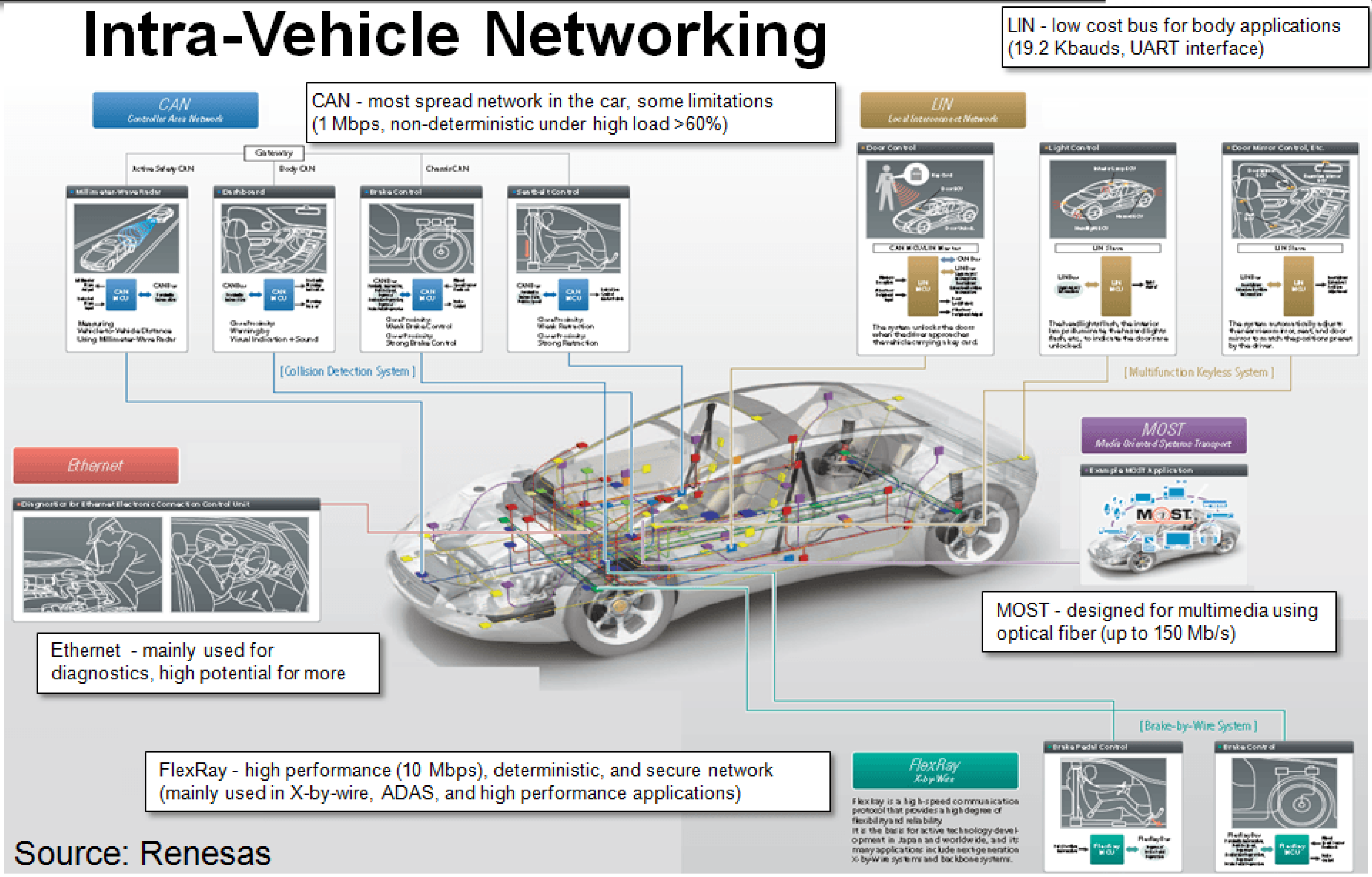
# Scalable Cyber Security from Endpoint to Enterprise

AKM Cyber, Corp.

## Designed for IoT

AKM was explicitly designed for internal automotive vehicle communication.




CAN
Controller Area Network
LIN
Local Interconnect Network
Gateway
[Collision Detection System]
[Multifunction Keyless System]
Ethernet
MOST
Media Oriented Systems Transport
FlexRay
X-by-Wire
[Brake-by-Wire System]

Intra-Vehicle Networking

LIN - low cost bus for body applications (19.2 Kbauds, UART interface)

CAN - most spread network in the car, some limitations (1 Mbps, non-deterministic under high load >60%)

Ethernet - mainly used for diagnostics, high potential for more

MOST - designed for multimedia using optical fiber (up to 150 Mb/s)

FlexRay - high performance (10 Mbps), deterministic, and secure network (mainly used in X-by-wire, ADAS, and high performance applications)

Source: Renesas

Automotive vehicle networks are a microcosm of typical, complex, mission and/or safety critical IoT closed systems.

## Origin Story

On November 1st, 2014, I met with a colleague who happened to be a 35+ year veteran of the automotive marketplace for one reason ….

I wanted to know his opinion of …

**Q: What was the #1 Problem in Automotive in 2014?**

**Hint: It is still the same problem in 2024!**

***The Answer: Security!***

## Origin Story

AKM was originally conceived as a solution for automotive, given that at the time (2014):

1) The automotive bus was an open architecture and unencrypted internally.
2) If encryption did exist, it was only for connection to a backend server of either the OEM or a service provider and the certificate was always static, for the life of the vehicle.
3) Every single proposed solution for inside the vehicle was a watered down implementation of PKI.

## Initial Goals for Autonomous Key Management

AKM CYBER CORP
AUTOMATING CYBERSECURITY

The goals we wanted to achieve with were simple:

1) Come up with a framework that simplifies the overall security process within a vehicle.
2) Eliminate the asymmetric key exchange.
3) Significantly reduce the overall costs of implementation and maintenance.
4) Enable security credentials that can be shared by two or more nodes to form a cryptographic trust relationship. Within an open bus architecture (such as is found in within automotive), this creates a virtual security mesh. Thus, enabling true, multipoint-to-multipoint, end-to-end, secure communication.
5) Refresh the security credentials on a frequent basis (which unfortunately is often times, never).
6) Unique Security Credentials to each logical AKM Trust Relationship (ATR) such that there is no association or relationship to security credentials in any other logical ATR. Meaning, hacking security credentials for one ATR will not provide insight into security credentials for other ATR.
7) Mitigate or Eliminate attacks found to commonly occur with PKI+TLS protected communication.

## Implementation Requirements of Solution

Those goals were then translated into the below requirements:

1) Given that the vehicle architecture is a closed IOT system (and all actors are known entities), simplify the authentication process. Thus, taking advantage of known actors, makes it possible for **replacing the asymmetric authentication phase** with a much simpler solution.
2) Each set of security credentials must be unique, with no association, either directly or indirectly, with any other cryptographic AKM Trust Relationship (ATR) and can be defined for cryptographic ATRs of two or more nodes.
3) **Authenticate** the **sender** without sharing any secrets, creating a Zero Knowledge Authentication solution (but, do so without the need for a 3rd party observer, as is required in classic ZKA solutions).
4) Reduce to **zero** if possible, any **latency** incurred during security session establishment.
5) **Refresh** security **credentials every** single **session**.
6) **Automate the security credential refresh process** while still maintaining uniqueness between sets of security credentials (hacking one set of security credentials, should not provide any information or insight that could lead to a breach of another set of security credentials).
7) **Eliminate** Man-in-the-Middle (**MITM**) **Attacks**.
8) **Eliminate Replay Attacks**.
9) Allow **sessions** to be **established** as both/either **event driven and/or time driven**.
10) **Do not reinvent the wheel**. Use existing standard cryptographic functions wherever possible.

## Our Technology

AKM CYBER CORP
AUTOMATING CYBERSECURITY

### Autonomous Key Management (AKM)

AKM is both a Decentralized, Distributed, Ledger-Based, Key Management System (KMS) and Multi-point communication security protocol layer.  **AKM can naturally act as a drop-in replacement for PKI + TLS**.

**AKM** uses a **Broadcast Architecture**, that supports **true, multi-point end-to-end encryption**.

The problem experienced by Zoom Video at the beginning of COVID in 2020 is a great example of the shortcomings of Point-to-Point PKI in a multi-point IoT environment.

# Future Vision

**Autonomous Key Management (AKM), solves all of the issues currently plaguing PKI and TLS implementations, thus:**

**AKM** will be the **preferred standard for IoT Security**, particularly in Industrial IoT, Smart City, Smart Home/Office, and Smart Factory environments.

**AKM** will become part of the **RFC set of protocols, replacing TLS** as the preferred security transport protocol.

**AKM will be the preferred method to secure groups of nodes on the battlefield and in remote areas like space and underwater** (both of which are difficult to "update" security credentials using PKI).

This is enabled as a consequence of AKM's ability to autonomously and independently, self-generate new security credentials, coupled with AKM's virtual security mesh concept that enables unique credentials to be assigned for each application over the same physical infrastructure with overlapping of security mesh groups across AKM Nodes.

# Implementations

## AKM ideally sits on top of UDP and/or TCP

The majority of implementations for AKM are on top of the TCP/IP Transport Layer, replacing TLS as the Transport Layer Security mechanism

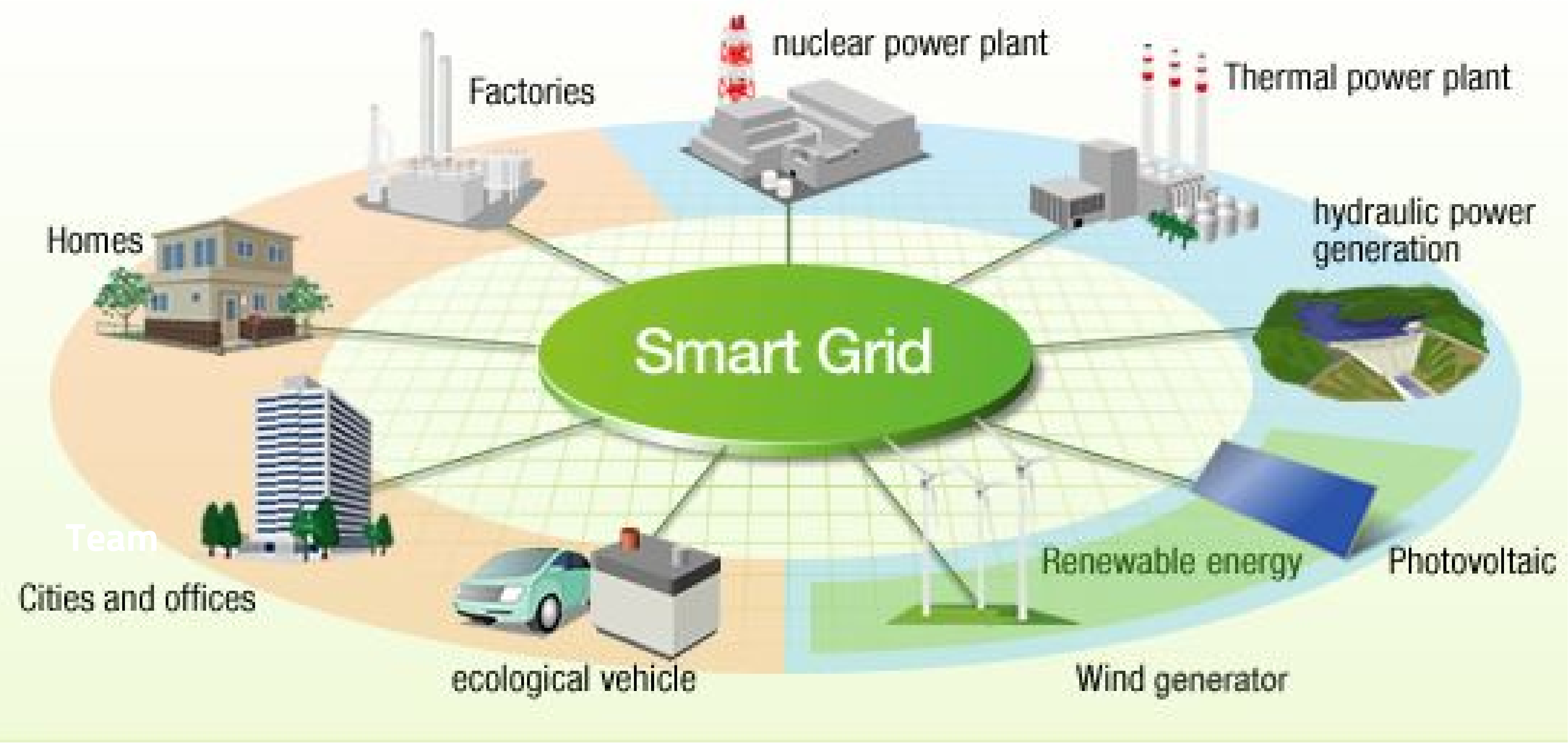
## AKM can be implemented directly on top of a physical layer driver

AKM has been implemented directly on top of a CAN driver, repurposing some of the CAN bits for AKM and using a proprietary, very thin transport layer on top of the CAN driver.

## AKM can be implemented within the MAC Layer (Layer-2)

AKM has been implemented within the MAC of a Deterministic IIoT Wireless Transceiver Technology as the MAC's Security Layer

# Security Framework Designed Explicitly for IoT



# AKM List of Features

Below is a list of AKM features and the underlying basis for how each feature is provided:

1) **Maintenance Free** – Once an AKM Trust Relationship (ATR) has been provisioned, no external maintenance should ever be required (thus, all ATRs are self-maintained and decentralized and require no external maintenance once provisioning has been completed).
2) **Real-time Data Analytics** – Because AKM is a protocol, Data Analytics are available on a per frame basis and thus, in real-time.
3) **Intrusion Detection** – Because Data Analytics can be constant and immediate, if enabled, intrusion detection is a built-in feature of AKM.
4) **Automatic Breach Recovery and Re-provisioning** – If enabled, because Intrusion Detection is a built-in feature, automatic re-provisioning of infected ATRs can be re-configured according to policy. Thus, effectively neutralizing threats as soon as a breach is discovered.
5) **Secure Boot with Device Authentication** – This feature uses the unique AKM Protocol Identifier associated with the device, in combination with an onboard, AKM enabled HSM or hardware secure element to provide a Secure Boot Feature to AKM enabled devices. Thus, ensuring only the precise associated host hardware is being used.
6) **Anti-spoofing and Network Authorization** – Because AKM can uniquely associate with a specific device, and because ATRs are constantly being automatically updated, stale or substitute devices cannot be re-inserted into an AKM protected network without being re-provisioned. This also ensures that only authorized devices can ever be inserted into an AKM protected network.

# AKM List of Features

7) **Replay Attack Protection** – The AKM protocol has a replay counter located within every frame. Thus, preventing previous frames from being retransmitted.

8) **Perfect Forward Secrecy** – Because Next Session Security Credentials are calculated based upon a randomly selected subset of parameters from the Parameter Data Vector (PDV), there is no mathematically available means to determine which parameters were used in prior sessions for calculating previous session Security Credentials.

9) **Security Credentials are Re-Generated and NOT Derived** – Because the Parameter Data Vector (PDV) is periodically updated based upon configured policy, next session credentials cannot be predicted subsequent to the replacement of the PDV, which is randomly generated and locally distributed within the de-centralized AKM Trust Relationship (ATR).

10) **Enterprise Grade Entropy** – Because the minimum PDV size is 128 and because the smallest allowable PDV subset is 15, assuming an evenly distributive function is used for selecting the PDV parameter subset, the entropy as calculated by the nPr function (the function which calculates the number of permutations of set size, 'n', and subset size, 'r'), is said to be: $1.72831541602 \times 10^{31}$. For reference the age of the universe as calculated in seconds is said to be somewhere between $10^{16}$ and $10^{17}$. Thus, even with a quantum computer, these parameters cannot be predicated without also knowing information that is never exposed (i.e. the internal seeds which are part of each set of AKM Security Credentials).

11) **Scalability at IoT Scale** – Because AKM may be configured as a broadcast architecture, and cryptographic Security Credentials can be configured for any number of 'n' nodes, where 'n' is any value greater than '1', there is no limitation with respect to the number of nodes or size of an ATR. An ATR is the entity for which the security credentials protect both data-in-flight and data-at-rest)

## AKM List of Features

12) **Unlimited AKM Trust Relationships (ATRs)** – Each ATR is unique from every other ATR. ATRs may be defined according to attributes and may co-exist with other virtual ATRs on the same AKM Node. Thus, communication can be performed securely on a per attribute basis.

13) **Low-Power and Energy Efficient** – Because only linear hashing functions and symmetric encryption is used, implementation of AKM requires minimum computational resources.

14) **Low-overhead** – Because Bulk Encryption begins with the very first frame of an AKM session, there is no appreciable latency (other than the protocol header) associated with an AKM protected network.

15) **Minimal Digital Footprint** – Edge Node AKM Software Applets are typically under 20K bytes and can be compressed down to below 10K bytes.

16) **True Multipoint End-to-End Encryption** – Cryptographic Security Credentials applied to ‘n’ nodes, where ‘n’ can be any number greater than or equal to 2.

17) **Quantum Resilient** – Because no public key is used (as in PKI with asymmetric encryption used for the key exchange, which in a closed IOT system is usually there for the lifetime of the system) and because AKM does not share any secrets, there is nothing for a quantum computing device to attack.

# AKM List of Features

18) **Limited Threat Surface** – This comes from two separate but related aspects of AKM. First, the only area within AKM that is truly "attackable" is where the secure credentials are stored. This is why, where possible, it is suggested to use a secure hardware storage device to secure the credentials. The second way that AKM limits the threat surface, is the fact that if a breach does occur in a node which does not use a secure hardware storage device, then, only the relationships contained within that breached node are affected. Because every single ATR is different from every other ATR, with no correlation of the security credentials of each other. They are completely distinct.

19) **Crypto-Agile** – Because AKM is a framework that allows for easy replacement of the cryptographic components (encryption algorithms, hash functions, parameter selection functions, block and key sizes, etc.), it is said to be, "crypto-agile".

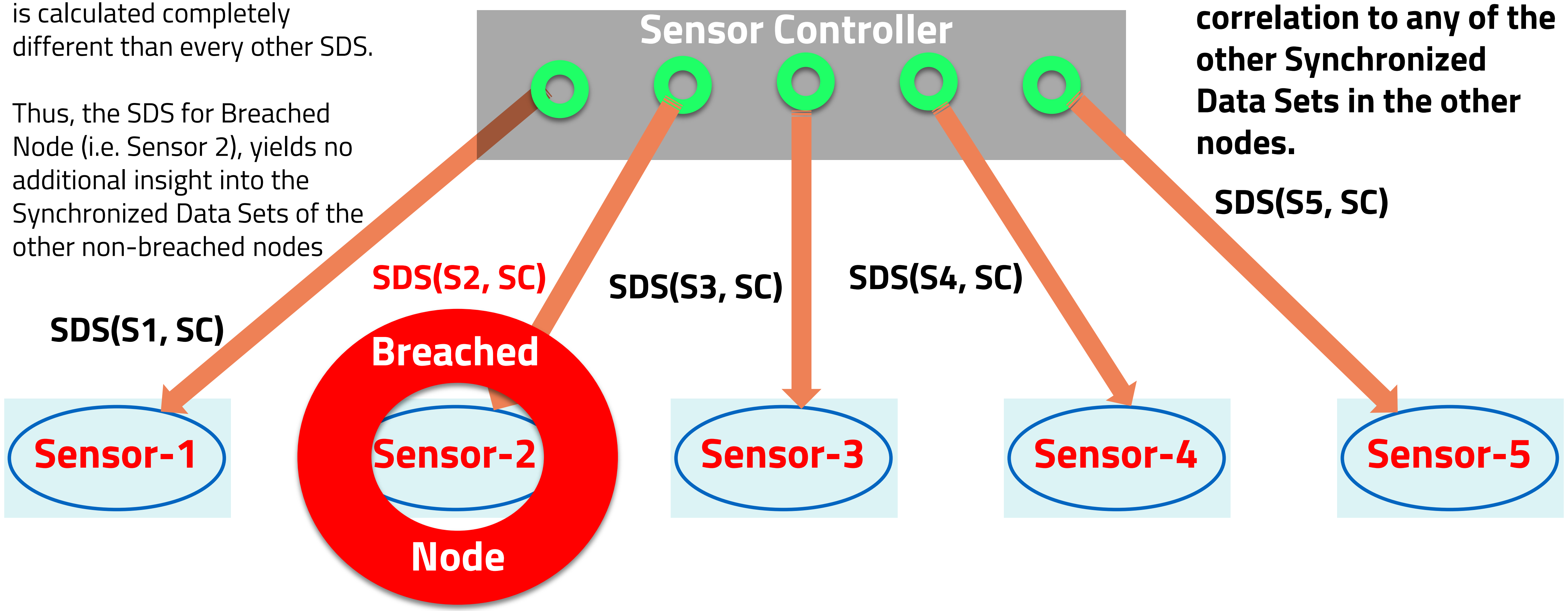
This system can operate for virtually any type of device, network configuration, and operating system. All that is needed for it to run is a minimalist transport layer. In the past, we have implemented a thin transport layer directly on top of a link-layer driver within a minimalist embedded RTOS.

# Limited Breach Potential

Each Synchronized Data Set (SDS) is unique and distinct and is calculated completely different than every other SDS.

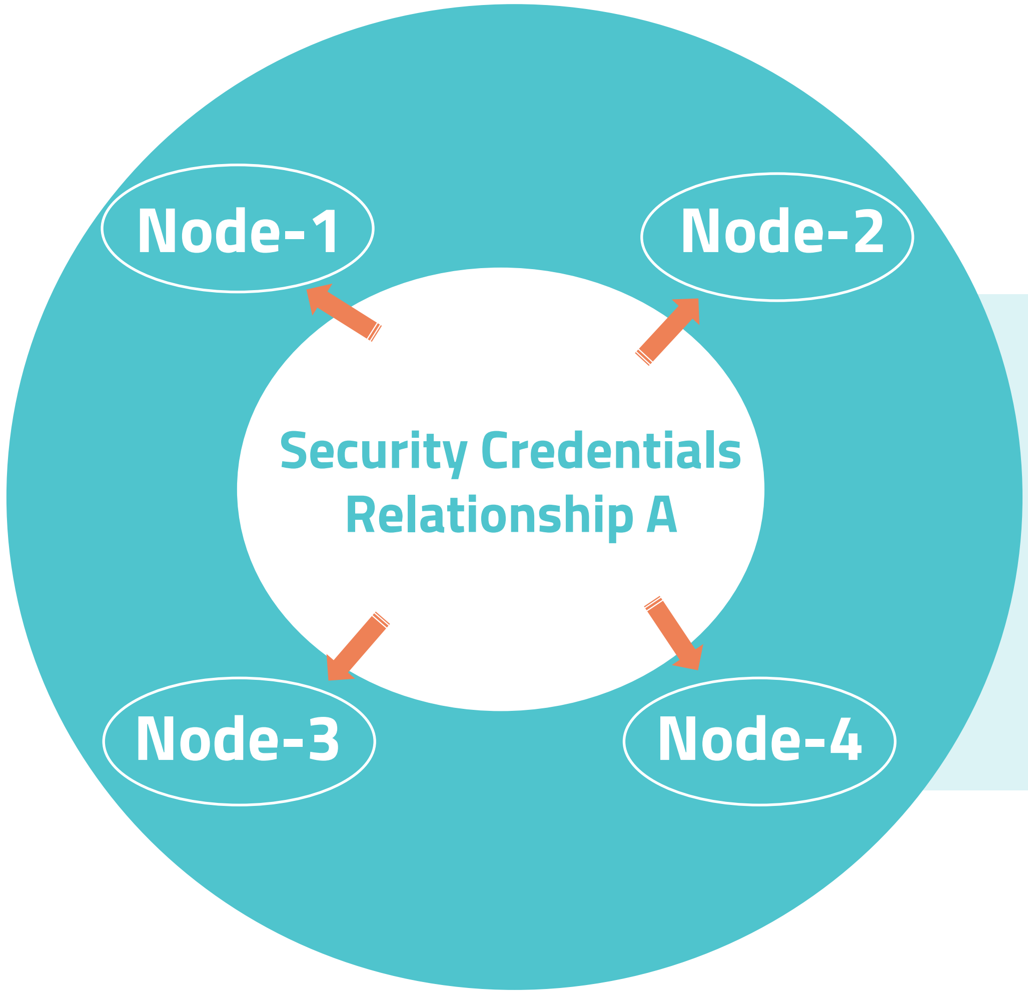
Thus, the SDS for Breached Node (i.e. Sensor 2), yields no additional insight into the Synchronized Data Sets of the other non-breached nodes

**SDS(S2, SC) has no correlation to any of the other Synchronized Data Sets in the other nodes.**



**A Breach is Implicitly Limited at its Point of Origin**

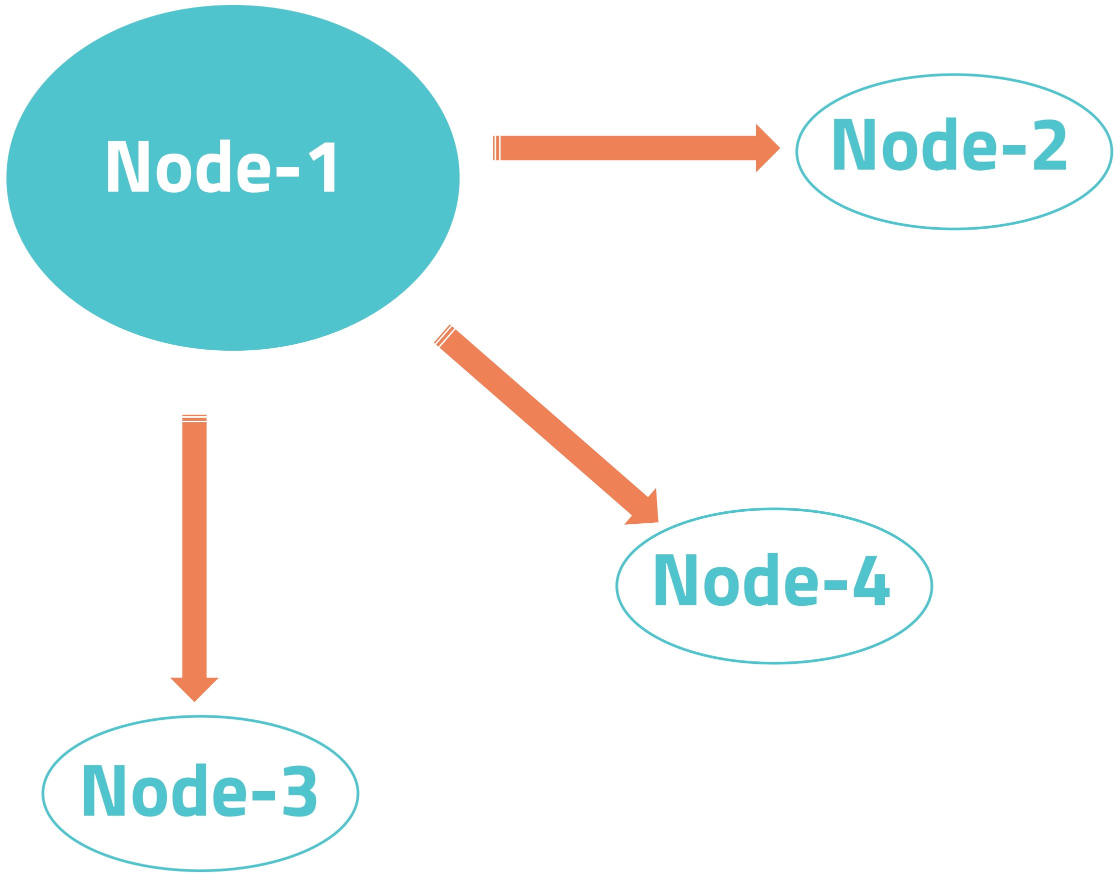
# Decentralized, Distributed Credentials



We are able to provide true, multi-point, end-to-end secure communication, because we use a decentralized, distributed architecture for maintaining and storing our credentials

**Decentralized, Distributive Ledger, KMS**

# Broadcast Architecture

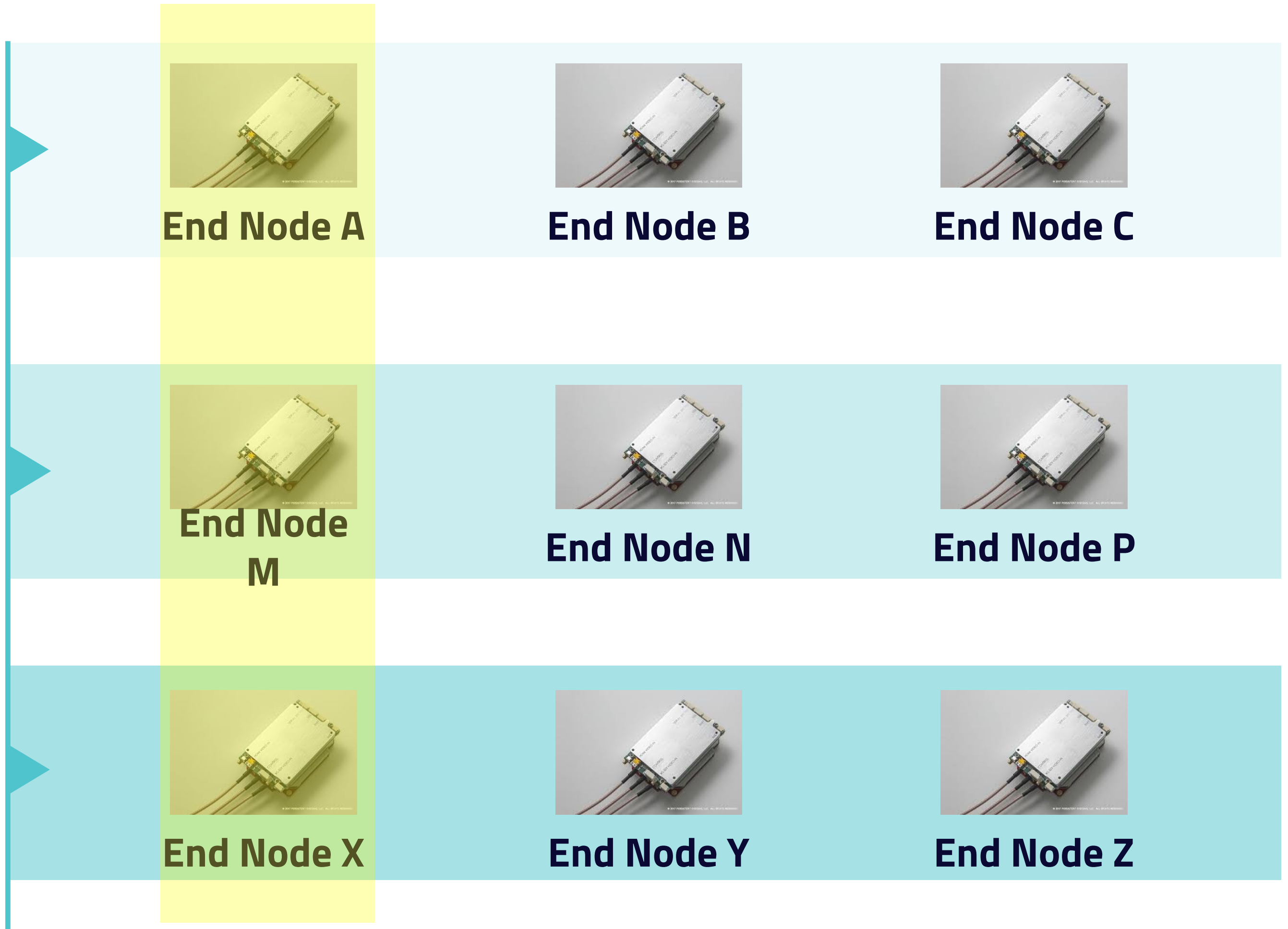


When nodes communicate with each other, they use the same set of credentials, which is how AKM supports true, multipoint end-to-end encryption.

Security Credentials are refreshed every session. Thus, unlike PKI, which in a closed IoT system, may never refresh its credentials or at best, do so every 9-12 months.

## Multi-point End-to-End Encryption

# Overlapping AKM Trust Relationships (ATRs) at the Application Layer

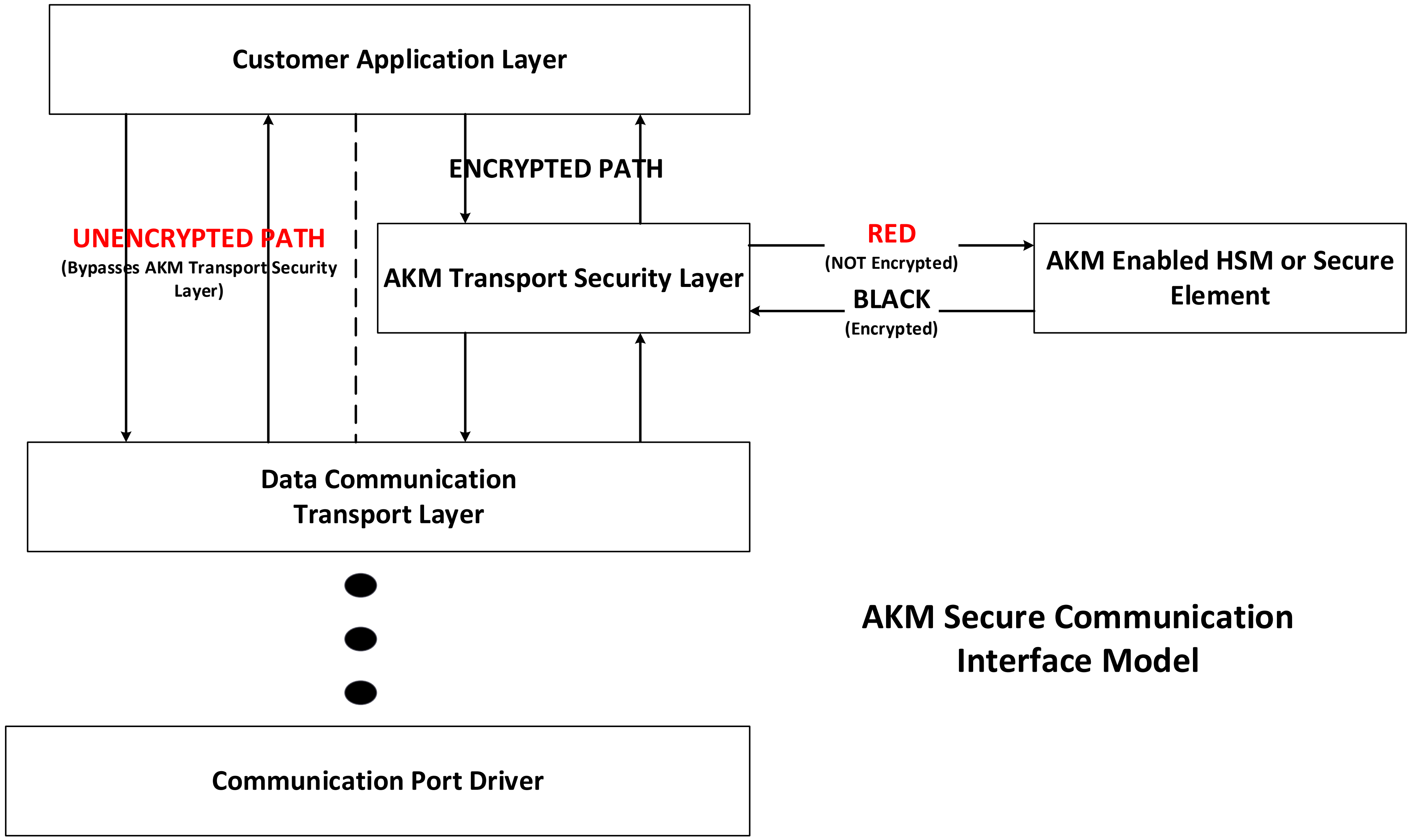


This diagram demonstrates that multiple AKM Trust Relationships (ATRs) can be held by the same nodes.

In this example, Nodes A, M, and X share an ATR, but each of them has a mutually exclusive second ATR with other nodes.

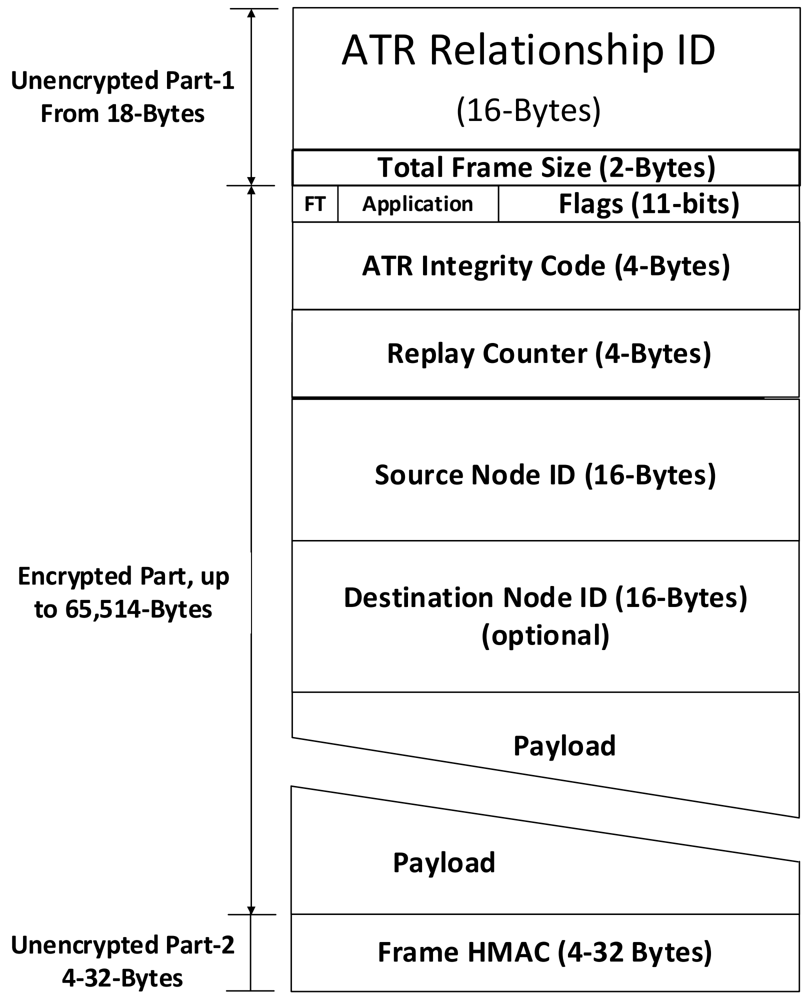
**Overlapping AKM Trust Relationships (One Ledger Per Relationship)**

# Transport Layer Security


Customer Application Layer
ENCRYPTED PATH
UNENCRYPTED PATH
(Bypasses AKM Transport Security Layer)
AKM Transport Security Layer
RED
(NOT Encrypted)
BLACK
(Encrypted)
AKM Enabled HSM or Secure Element
Data Communication Transport Layer
Communication Port Driver

AKM Secure Communication Interface Model

# Features and how they are implemented



Unencrypted Part-1 From 18-Bytes
ATR Relationship ID (16-Bytes)
Total Frame Size (2-Bytes)
FT
Application
Flags (11-bits)
ATR Integrity Code (4-Bytes)
Replay Counter (4-Bytes)
Source Node ID (16-Bytes)
Encrypted Part, up to 65,514-Bytes
Destination Node ID (16-Bytes) (optional)
Payload
Payload
Unencrypted Part-2 4-32-Bytes
Frame HMAC (4-32 Bytes)

## Unencrypted Part-1

RelationshipID – The Relationship Identifier is unencrypted and enables the Broadcast Architecture default behavior of AKM.

Total Frame Size – Size of the frame, including the Frame HMAC.

## Encrypted Part

Frame Type Bit – This bit determines if the frame is a Data Frame, or a Command Frame (including Session Status reporting and/or queries, Integrity Management, or Data Analytics information).

Application – Optional field specifying a particular device application.

Flags – These bits indicate information about the frame, including whether or not there is an optional Destination Node Identifier field, and enable data frames to convey status and state information about the relationship, the node, and/or the frame and thus decrease the need for explicit command frames.

ATR Integrity Code – used to provide implicit zero trust functionality, given that every AKM Endpoint has previously authenticated both its endpoint and its membership within an AKM Trust Relationship

Replay Counter – Replay Counter increments on every frame.

SourceNodeID – This identifies the sender of the frame.

DestinationNodeID – This optional field identifies the target recipient, if any, of the frame

Payload – This is the encapsulated payload, representing the actual data being communicated.

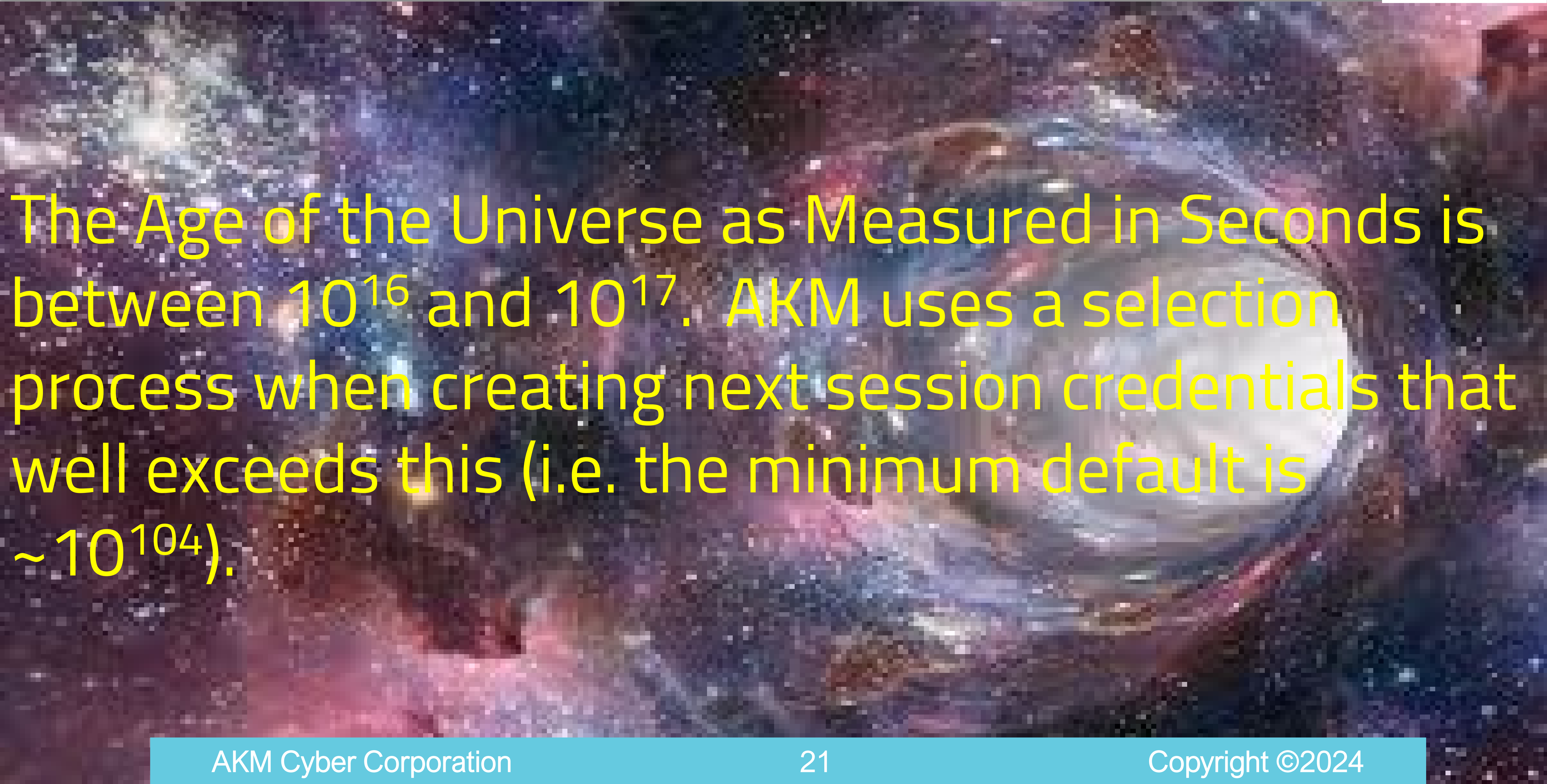
## Unencrypted Part-2

Frame HMAC – This is a SHA256 (or SHA512) based HMAC, that digitally authenticates the entire frame, including the unencrypted part of the frame header.

AKM uses an EtM scheme, which encrypts and then does the HMAC. As reported by Y-Combinator Hacker News, “The recommended, safer, and sturdier method is EtM (Encrypt then MAC), in which you encrypt the plaintext then MAC the ciphertext. This way you can verify the cipher before you operate on it. EtM is [according to Wikipedia] "[...] the only method which can reach the highest definition of security in AE.*

* https://news.ycombinator.com/item?id=15950481

# Entropy of Security Credentials



The Age of the Universe as Measured in Seconds is between $10^{16}$ and $10^{17}$. AKM uses a selection process when creating next session credentials that well exceeds this (i.e. the minimum default is ~$10^{104}$).

## Basis for how the Entropy is Calculated

| nPr Function | | |
|---|---|---|
| r-Subset Size | n-Set Size | P-Number of Permutations |
| 15 | 128 | 1.72832E+31 |
| 20 | 128 | 2.91113E+41 |
| 25 | 128 | 3.89402E+51 |
| 15 | 256 | 8.75002E+35 |
| 20 | 256 | 6.82277E+47 |
| 25 | 256 | 4.78631E+59 |
| 15 | 512 | 3.54105E+40 |
| 20 | 512 | 1.05232E+54 |
| 25 | 512 | 2.97249E+67 |
| | | |
| | | |
| 25 | 1024 | 1.3466E+75 |

# How the Parameter Data Vector (PDV) is used

The AKM Parameter Data Vector (PDV) is an implementation specific vector that can be used across all nodes within an ATR, that in combination with seed values from the Synchronized Data Set (SDS), is used to create the cryptographic security credentials for the next session for a specific ATR.

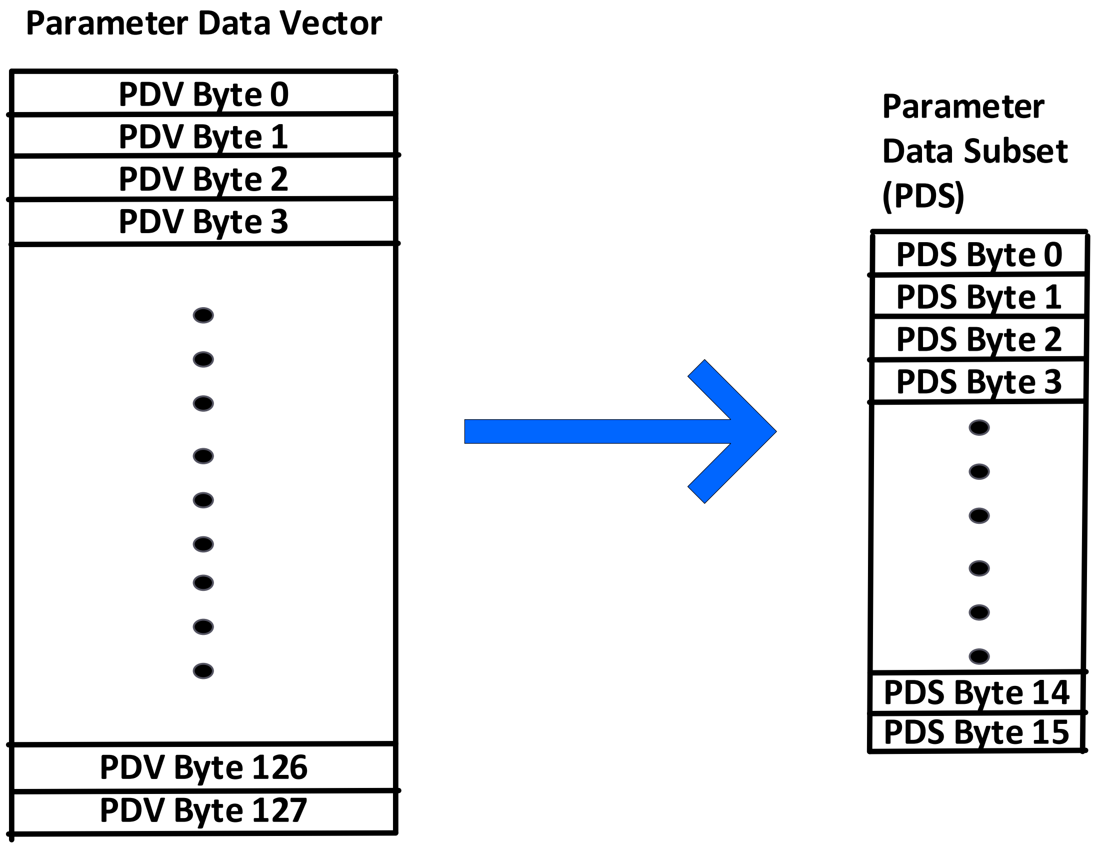
There should be a unique PDV per ATR. Further, although, for reasons of simplicity, it is strongly suggested not to change this on a per session basis, as it is unnecessary given the nature of how all values that could possibly lead to a breach are never shared externally from where they are calculated.

The minimum size of the PDV is 128-bytes, with each element within the PDV being 1-byte each, so there are 128-values. However, for implementations requiring military grade security, it is suggested to use a PDV of 512-bytes and a subset size of 52-bytes. This would yield an equivalent entropy level of $10^{139}$. The subset of 52 is chosen, because, the algorithm will be selecting 16-bytes for the next session symmetric key, 32-bytes for the next session HMAC Secret, and 4-bytes for the next session seed. Note, the seed value will update as a consequence of every session refresh and do so in synch across all other nodes within an ATR.

The PDV should be regenerated on a semi-periodic basis and distributed across all nodes within the ATR. It is suggested that the new PDV use a True Random Number Generator (TRNG) source to ensure true randomness when generating the next PDV.

# Derivation of SDS from PDV using the Fisher-Yates Shuffle

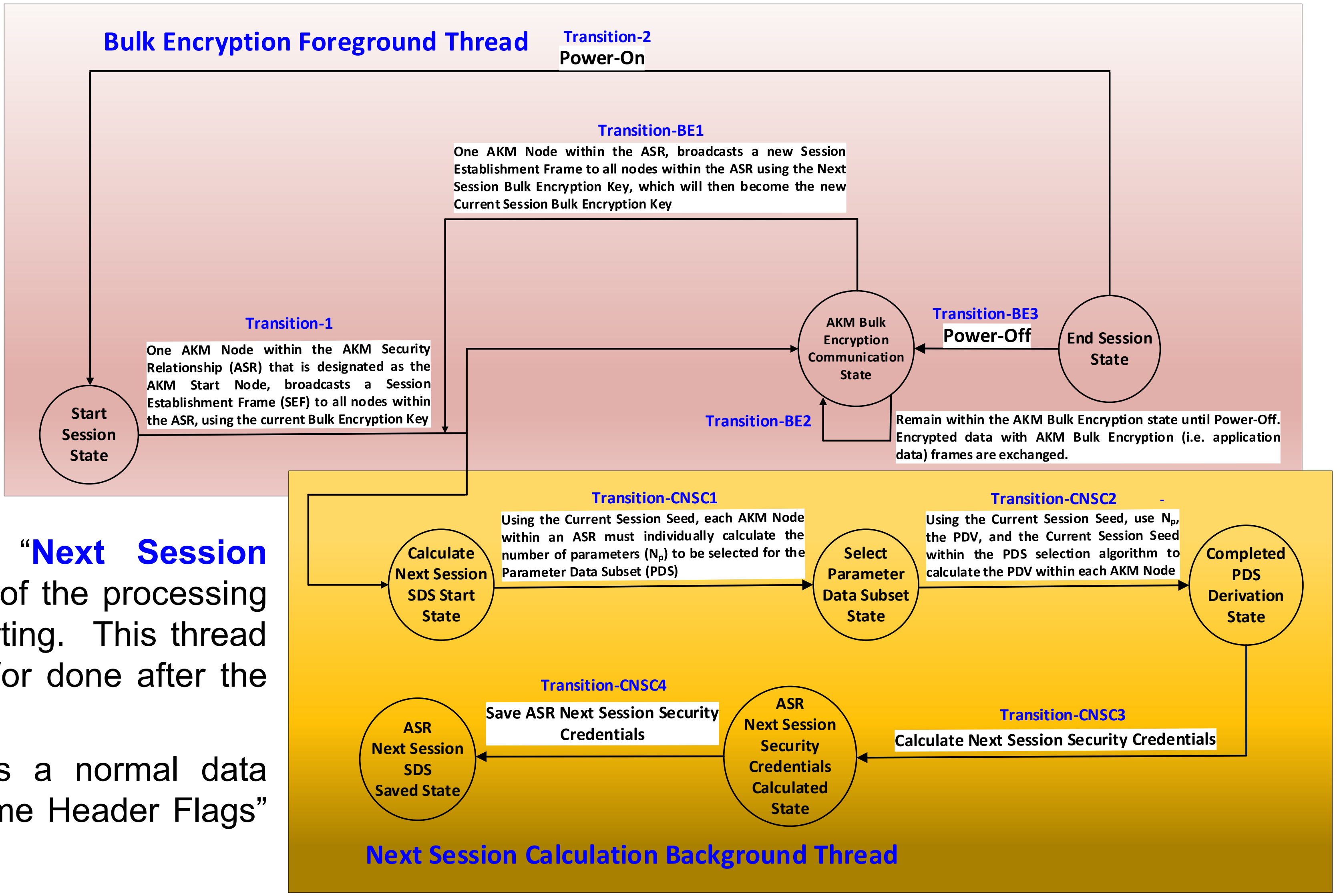
***ParameterSelectionFunction (PDV)***



In going from the PDV to creation of the SDS, the AKM framework must first select the Parameter Data Subset (PDS) from which the parameters will be used to create Next Session Credentials.

# Derivation of SDS from PDV via the PDS

As illustrated in the diagram, there are **two parallel paths that are executed simultaneously**. One is a foreground thread and the other is a background thread. The foreground thread, which is the "**Bulk Encryption Thread**", is responsible for the processing of the actual bulk encryption data. This is the thread in which the actual application data is exchanged and thus encrypted data going out and decrypted data coming in.


Bulk Encryption Foreground Thread
Transition-2
Power-On
Transition-BE1
Transition-1
Start Session State
AKM Bulk Encryption Communication State
Transition-BE3
Power-Off
End Session State
Transition-BE2
Transition-CNSC1
Transition-CNSC2
Calculate Next Session SDS Start State
Select Parameter Data Subset State
Completed PDS Derivation State
Transition-CNSC4
Save ASR Next Session Security Credentials
Transition-CNSC3
Calculate Next Session Security Credentials
ASR Next Session SDS Saved State
ASR Next Session Security Credentials Calculated State
Next Session Calculation Background Thread

Transition-BE1: One AKM Node within the ASR, broadcasts a new Session Establishment Frame to all nodes within the ASR using the Next Session Bulk Encryption Key, which will then become the new Current Session Bulk Encryption Key

Transition-1: One AKM Node within the AKM Security Relationship (ASR) that is designated as the AKM Start Node, broadcasts a Session Establishment Frame (SEF) to all nodes within the ASR, using the current Bulk Encryption Key

Transition-BE2: Remain within the AKM Bulk Encryption state until Power-Off. Encrypted data with AKM Bulk Encryption (i.e. application data) frames are exchanged.

Transition-CNSC1: Using the Current Session Seed, each AKM Node within an ASR must individually calculate the number of parameters ($N_p$) to be selected for the Parameter Data Subset (PDS)

Transition-CNSC2: Using the Current Session Seed, use $N_p$, the PDV, and the Current Session Seed within the PDS selection algorithm to calculate the PDV within each AKM Node

The background thread, which is the "**Next Session Calculation Thread**", is responsible for all of the processing that must occur prior to a new session starting. This thread can either be done in the background and/or done after the current session closes.

**NOTE:** A Session Establishment frame is a normal data frame, but with the SEF bit within the "Frame Header Flags" field set.

# Interchangeable, Drop In Functions

All of the “drop-in functions” that need to be specified as part of the “crypto-agile” implementation are as follows:

- ❑ The **Encryption Algorithm** used for **encrypting and decrypting** data
- ❑ The **Cryptographic Hash Function –** used in conjunction with **the HMAC**
- ❑ The **Hash-based Message Authentication Code (HMAC) algorithm –** used for calculating the **frame HMAC**
- ❑ The **Cryptographic Hash Function** used in conjunction with the next Session Seed for calculating the size of the subset to be selected from the Parameter Data Vector (PDV)
- ❑ A “seeded” **Pseudo Random Number Generator” –** used to create random numbers as part of **the Parameter Selection Function** in selecting evenly distributed numbers from the PDV.
- ❑ The **Parameter Selection Function** used in conjunction with the **next Session Seed** for selecting the parameters from the PDV
- ❑ The **Cryptographic Hash Function** used for calculating the **next Session MAC seed**
- ❑ The **Cryptographic Hash Function** used for calculating the **next Session Encryption Key**
- ❑ The **Cryptographic Hash Function** used for calculating the **next Session Seed**

# Default Values of the AKM "Crypto-Agile" Framework

AKM is a framework, that is both a cryptographic Key Management System (KMS) and a Secure Communication Protocol. The default elements of this framework are:

- The Encryption Protocol – The default selection is AES128
- The Hash Function – The default selection is SHA256
- The Parameter Selection Function – The default selection is the Fisher-Yates Shuffle (or the more modern name, the Knuth Shuffle)
- The Parameter Data Vector (PDV) Size – The default size is 512
- The Parameter Data Set (PDS) Size – The default size is 52
- Session Renewal Timer – The default timeout is 24-hours
- Parameter Data Vector Renewal Timer – The default timeout is 7-days
- Maximum Security Group Size – The default maximum number of nodes is 20

All of the algorithms may be swapped out for other algorithms that perform the same functionality. All of the size and timeout values may also be adjusted down or up without affecting the integrity of the framework. The framework is intended to be as flexible as possible.

## The AKM Trust Relationship (ATR) Representative – The SDS

AKM CYBER CORP
AUTOMATING CYBERSECURITY

- The AKM Synchronized Data Set (SDS) represents the set of security credentials for a specific AKM Trust Relationship.
- This is a set of values, one per AKM Trust Relationship (ATR), used to maintain synchronization within the group across sessions.
- The SDS Next Session Seed Values are the most important elements within each set.
- These seed values, combined with the associated commonly shared algorithms, determines which parameters from the Parameter Data Vector are selected for the forthcoming session.

## Common Elements of an SDS

Elements within the SDS, include, but are not limited to:

- Relationship Identifier
- Current Session Encryption Key;
- Current Session Seed Value;
- Current Session HMAC Secret used for HMAC calculations;
- Next Session Encryption Key;
- Next Session Seed Value;
- Next Session HMAC Secret used for HMAC calculations;
- Optional Arbiter Session Security Credentials;
- Fallback Session Security Credentials;
- Failsafe Session Security Credentials;
- AKM protocol addresses of all of the AKM nodes that are part of the same AKM Trust Relationship (ATR).

# Common Elements of an SDS (Example Data Structure)

```
typedef struct _SDS_PARAMETERS {      // SizeOf (404-bytes)
    AKM_SECURITY_RELATIONSHIP_IDENTIFIER      ASR_Security_Relationship_ID;           // Offset:   0 -  15 ( 16-bytes)

    AES_128_BIT_ENCRYPTION_KEY                Current_Session_Encryption_Key;         // Offset:  16 -  31 ( 16-bytes)
    U32                                       Current_Session_Seed_Value;             // Offset:  32 -  35 (  4-bytes)
    HMAC_KEY                                  Current_Session_HMAC_Key;               // Offset:  36 -  67 ( 32-bytes)
    AES_128_BIT_ENCRYPTION_KEY                Next_Session_Encryption_Key;            // Offset:  68 -  83 ( 16-bytes)
    U32                                       Next_Session_Seed_Value;                // Offset:  84 -  87 (  4-bytes)
    HMAC_KEY                                  Next_Session_HMAC_Key;                  // Offset:  88 - 119 ( 32-bytes)
    AES_128_BIT_ENCRYPTION_KEY                Fallback_Recovery_Encryption_Key;       // Offset: 120 - 135 ( 16-bytes)
    U32                                       Fallback_Recovery_Seed_Value;           // Offset: 136 - 139 (  4-bytes)
    HMAC_KEY                                  Fallback_Recovery_HMAC_Key;             // Offset: 140 - 171 ( 32-bytes)
    AES_128_BIT_ENCRYPTION_KEY                Failsafe_Recovery_Encryption_Key;       // Offset: 172 - 187 ( 16-bytes)
    U32                                       Failsafe_Recovery_Seed_Value;           // Offset: 188 - 191 (  4-bytes)
    HMAC_KEY                                  Failsafe_Recovery_HMAC_Key;             // Offset: 192 - 223 ( 32-bytes)

    // Optional Arbiter Session Security Credentials -- used for dynamic provisioning
    AES_128_BIT_ENCRYPTION_KEY                Arbiter_Session_Encryption_Key;         // Offset: 224 - 239 ( 16-bytes)
    U32                                       Arbiter_Session_Seed_Value;             // Offset: 240 - 243 (  4-bytes)
    HMAC_KEY                                  Arbiter_Session_HMAC_Key;               // Offset: 244 - 275 ( 32-bytes)

    // Parameter Data Vector (PDV)
    PROGRAM_DATA_VECTOR                       Parameter_Data_Vector;                  // Offset: 276 - 403 (128-bytes) */
} SDS_PARAMETERS, *PSDS_PARAMETERS;
```

# Common Elements of an SDS (Example Data Structure)

```
// This structure represents a single Synchronized Data Set (SDS) for physical device AKM provisioning modules
// and is always organized from the perspective of the chain of trust.  The AKM Object representing the
// AKM Provisioner is always higher on the chain of trust than the object representing the AKM Provisionee.

typedef struct _ATR_PHYSICAL_DEVICE_PROVISIONING_SYNCHRONIZED_DATA_SET {
   SDS_PARAMETERS                          SDS_parameters;                      // Offset:   0 - 403 (404-bytes)

   // As of this writing, this should be one of the following values:
   //
   //   Ultimate_AKM_Root_Of_Trust_Backend_Server_object,
   //   Intermediate_AKM_Chain_of_Trust_Proxy_Server_object,
   //   Local_AKM_Chain_of_Trust_management_module_object,
   //   Portable_AKM_Chain_of_Trust_provisioning_device_object,
   ATR_OBJECT_DELINEATOR_ENUM              AKM_Provisioner_Object_Delineator;   // Offset: 404 - 407 ( 4-bytes)
   AKM_OBJECT_STANDARD_128_BIT_ADDRESS     Provisioning_Module_AKM_Address;     // Offset: 408 - 423 (16-bytes)

   // As of this writing, this can be one of the following values:
   //
   //   Intermediate_AKM_Chain_of_Trust_Proxy_Server_object,
   //   Local_AKM_Chain_of_Trust_management_module_object,
   //   Portable_AKM_Chain_of_Trust_provisioning_device_object,
   //   AKM_Communication_Edge_Node_object
   ATR_OBJECT_DELINEATOR_ENUM              AKM_Provisionee_Object_Delineator;   // Offset: 424 - 427 ( 4-bytes)
   AKM_OBJECT_STANDARD_128_BIT_ADDRESS     Subservient_Object_AKM_Address;      // Offset: 428 - 443 (16-bytes)

} ATR_PHYSICAL_DEVICE_PROVISIONING_SYNCHRONIZED_DATA_SET, *PATR_PHYSICAL_DEVICE_PROVISIONING_SYNCHRONIZED_DATA_SET;
```

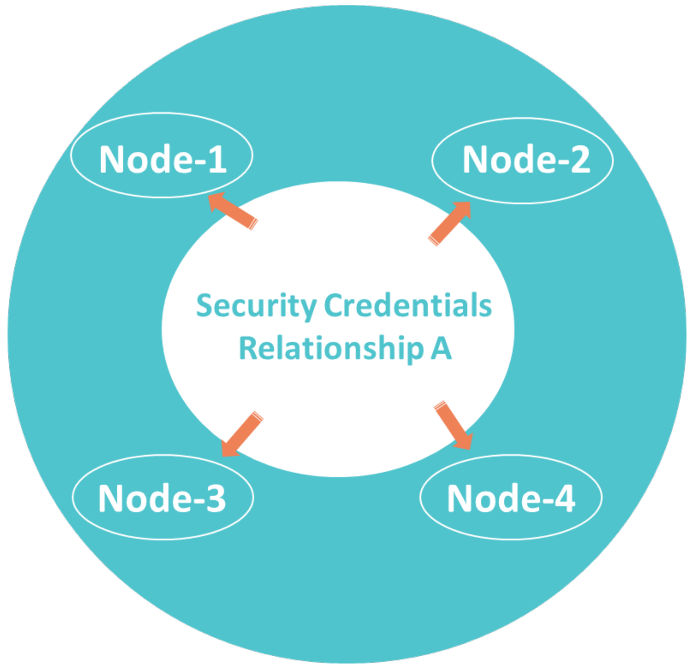
# The Synchronized Data Set Table (i.e. the AKM Ledger)

**SDS Table**

| | |
|---|---|
| **Factory Provisioning Relationship** | OEM Factory Server AKM Provisioning Relationship SDS |
| **BackOffice Server Provisioning Relationship** | BackOffice Server AKM Provisioning Relationship SDS |
| **Field Provisioning Unit Provisioning Relationship** | Field Provisioning Unit AKM Provisioning Relationship SDS |
| | In-Network Management Device Provisioning Relationship SDS |
| **AKM Relationship-1** | AKM Peer-to-Peer Relationship SDS (1) |
| **AKM Relationship-2** | AKM Peer-to-Peer Relationship SDS (2) |
| | ● ● ● |
| **AKM Relationship-n-1** | AKM Peer-to-Peer Relationship SDS (n-1) |
| **AKM Relationship-n** | AKM Peer-to-Peer Relationship SDS (n) |

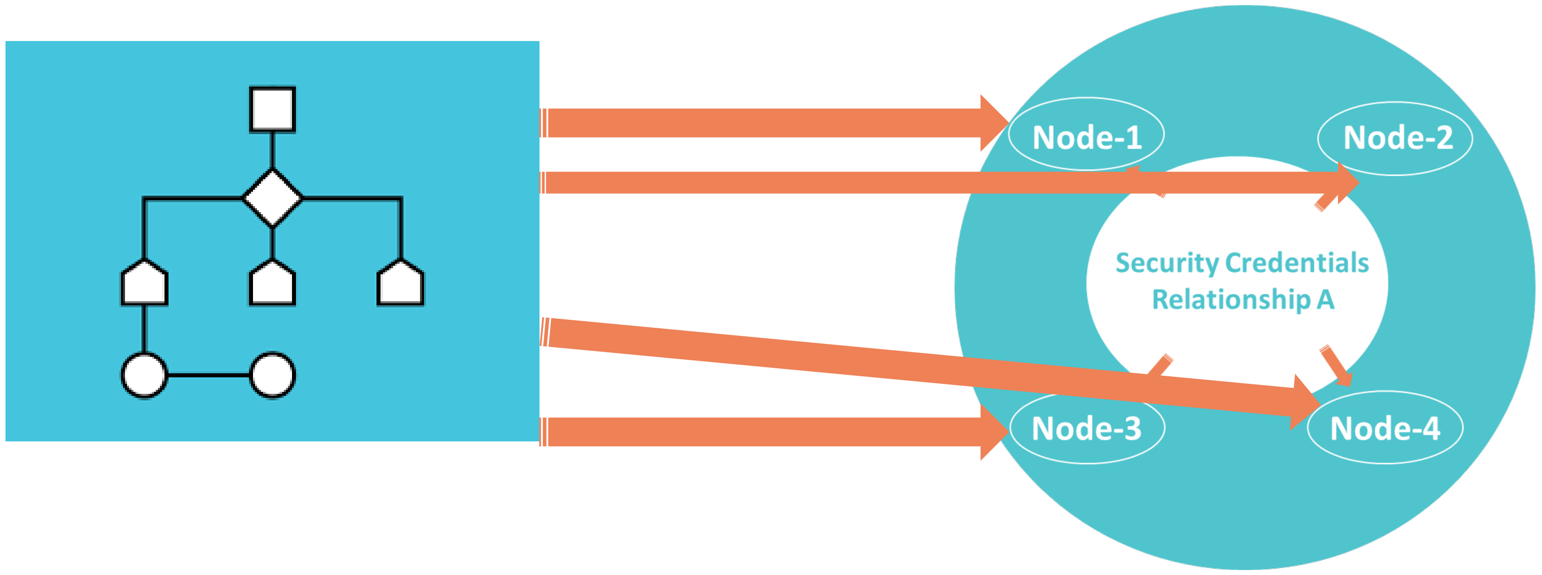
Every entry within the SDS Table (or AKM Ledger) represents a different SDS (meaning, a different AKM Trust Relationship).

## Same SDS Entry in All Nodes within an AKM Trust Relationship



Multi-point, end-to-end encryption is achieved by embedding the same security credentials within each AKM Node (endpoint) of the same AKM Trust Relationship (ATR).

## Common Algorithms Execute on the Same Data



AKM uses the same algorithms on the same data to re-create the same set of next session credentials on every node within the same AKM Trust Relationship (ATR).

## Synchronization across Security Sessions

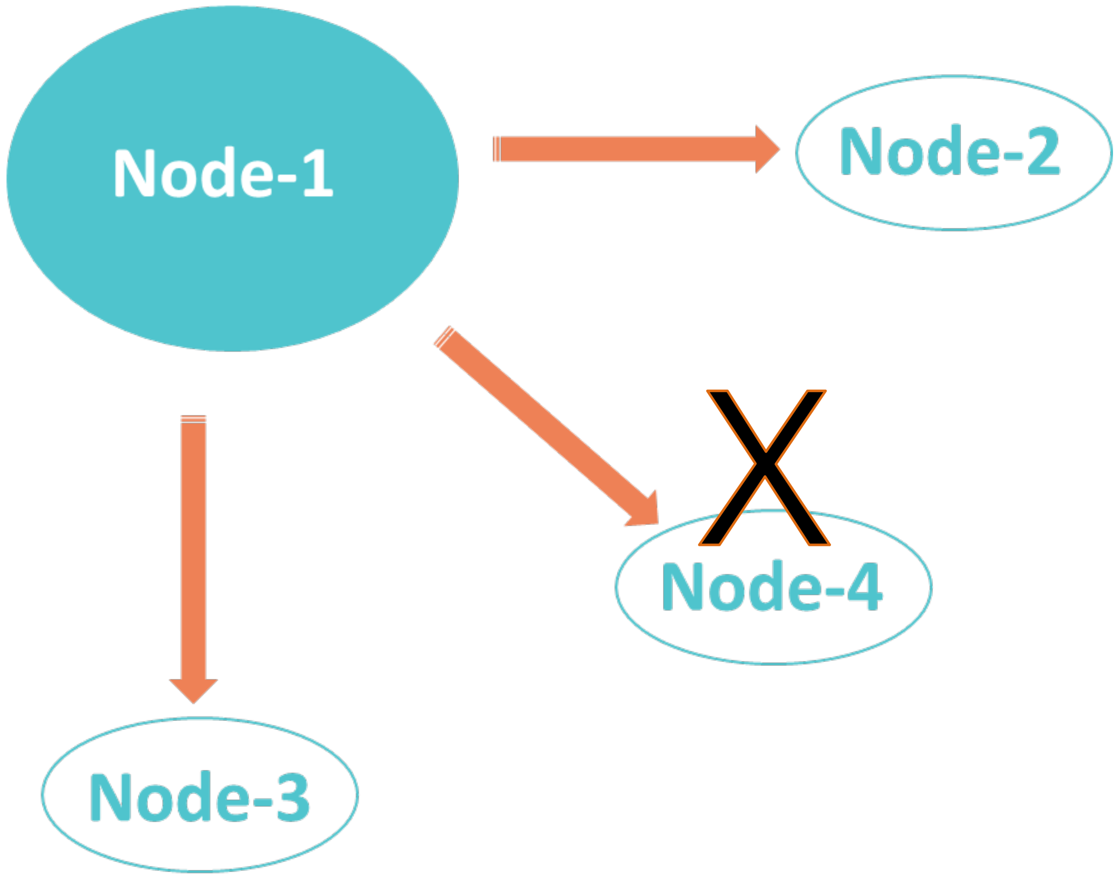
AKM does this by maintaining synchronization between all nodes within the same ATR and triggering off of the same event and/or time-out.

## Simple to Use because credentials are Autonomously Managed

Simple to Understand: The basic premise is simple, use the same algorithms on the same data and expect the same results.

Simple to Use: Configure the AKM Trust Relationship (ATR) once and never touch the ATR again. **One and Done!**

# Broadcast Architecture in Peer-to-Peer Security Mesh



Because AKM employs a connectionless, virtual Broadcast Mesh Architecture and the same credentials are distributed to every endpoint within the AKM Trust Relationship (ATR), a node can drop out of the ATR in real-time without affecting the ability of the other AKM Nodes to maintain communication and synchronization.

## Reduced Labor Requirements

AKM CYBER CORP
AUTOMATING CYBERSECURITY

PKI Requires Knowledgeable Full-time, Certified, Professional Security Staff to install and manage.

AKM requires only knowing which nodes you wish to group together into AKM Trust Relationships (ATRs).

## Longevity of Solution, because AKM is Future Ready

Asymmetric Encryption, which is the cornerstone of PKI will be broken as soon as commercially available Quantum Devices are available.

AKM never shares any knowledge, encrypted or otherwise, that could be used by an unwanted observer to breach a network. Thus, nothing for a Quantum Computing Device to attack!

## The Coming Cryptographic Apocalypse

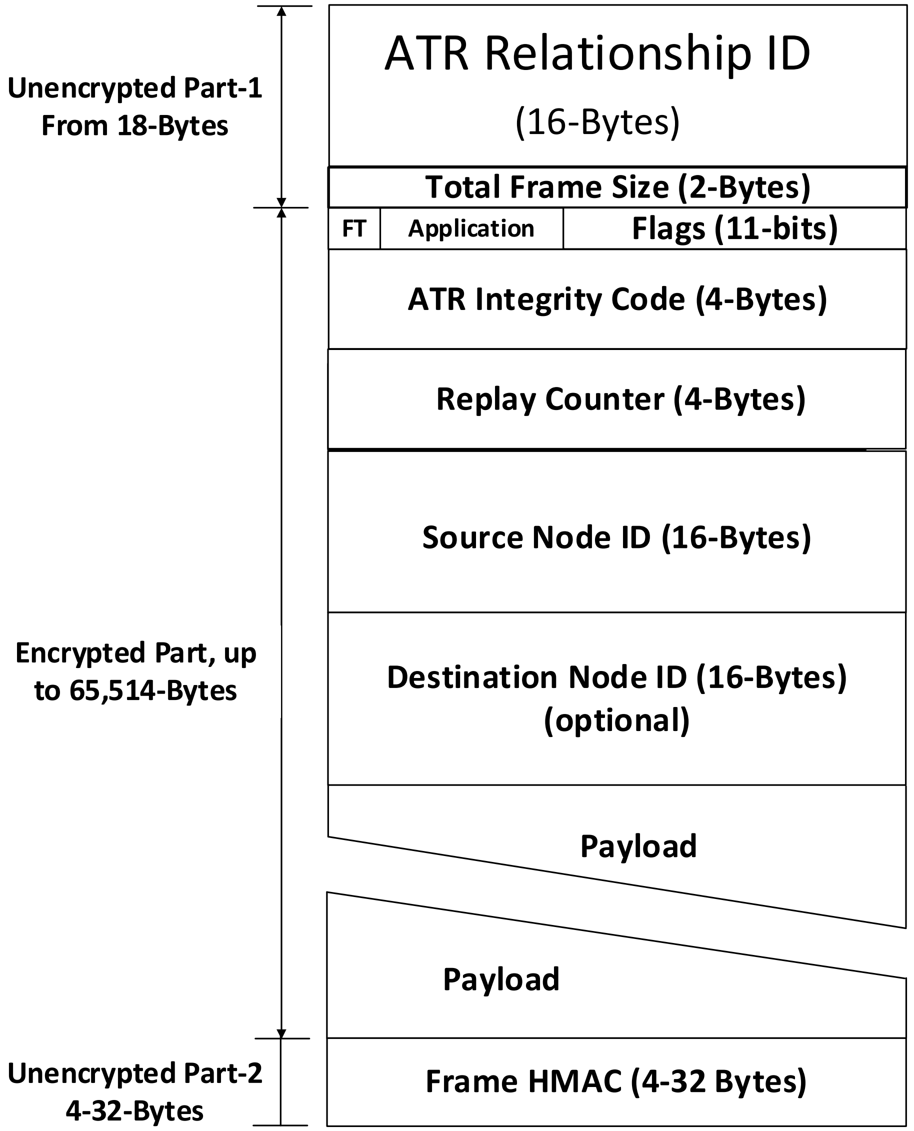
*Quantum computers will likely soon break traditional public key cryptography, including the ciphers protecting most of the world's digital secrets. These soon-to-be-broken protocols and components include HTTPS, TLS, SSH, PKI, digital certificates, RSA, DH, ECC, most Wi-Fi networks, most VPNs, smartcards, HSMs, most cryptocurrencies, and most multifactor authentication devices that rely on public key crypto. If the list just included HTTPS and TLS, it would cover most of the Internet. On the day that quantum computing breaks traditional public crypto, every captured secret protected by those protocols and mechanisms will be readable.*

*Grimes, Roger A.. Cryptography Apocalypse (p. xxii), Wiley, Kindle Edition, November 12th, 2019 .*

# No MITM, Nothing to Observe that can lead to a breach

Five key elements of AKM offer protection against MITM attacks, including protection from spoofing and replay attacks:

1) The ATR Integrity Code, which is common to all AKM Endpoints within an ATR and provides an implicit Zero Trust token mechanism.
2) The presence of the "Replay Counter" ensures against repurposing previously sent frames and make it difficult to send modified imposter frames.
3) The "Frame Header HMAC Authentication" value authenticates the entire frame, including the unencrypted parts of the frame header. Thus, ensuring the frame header is not compromised.
4) The "Replay Counter" is both encrypted and protected by the Frame Header Digital Signature, hence, preventing modification of it
5) There are no shared secrets that are ever exchanged or exposed that would allow an attacker insight into the AKM Security Credentials.


Unencrypted Part-1 From 18-Bytes
ATR Relationship ID (16-Bytes)
Total Frame Size (2-Bytes)
FT
Application
Flags (11-bits)
ATR Integrity Code (4-Bytes)
Replay Counter (4-Bytes)
Source Node ID (16-Bytes)
Encrypted Part, up to 65,514-Bytes
Destination Node ID (16-Bytes) (optional)
Payload
Payload
Unencrypted Part-2 4-32-Bytes
Frame HMAC (4-32 Bytes)

# Next Session Credentials, created with Perfect Forward Secrecy

AKM CYBER CORP
AUTOMATING CYBERSECURITY

AKM Next Session Credentials are created using a randomized subset of parameters from the Parameter Data Vector (PDV). Assuming the implementation is correct, including ensuring the parameter selection function used has a equally distributive selection process, then, the likelihood of someone guessing what the selected parameters were, including their order of selection can be calculated by the nPr function.

**nPr Function**

| r-Subset Size | n-Set Size | P-Number of Permutations |
|---|---|---|
| 15 | 128 | 1.72832E+31 |
| 20 | 128 | 2.91113E+41 |
| 25 | 128 | 3.89402E+51 |
| 15 | 256 | 8.75002E+35 |
| 20 | 256 | 6.82277E+47 |
| 25 | 256 | 4.78631E+59 |
| 15 | 512 | 3.54105E+40 |
| 20 | 512 | 1.05232E+54 |
| 25 | 512 | 2.97249E+67 |
| | | |
| | | |
| 25 | 1024 | 1.3466E+75 |

# Local Chain of Trust

The **OEM Factory Server** & Configuration Module represents the OEMs original database settings. The customer may or may not wish the OEM to maintain its link to the field equipment. However, most OEMS will probably keep at least one AKM Provisioning Relationship for recalls and maintenance.

The **Customer BackOffice Server** is the practical top-level chain-of-trust, since it must maintain provisioning relationships for all fielded equipment within its operational domain.

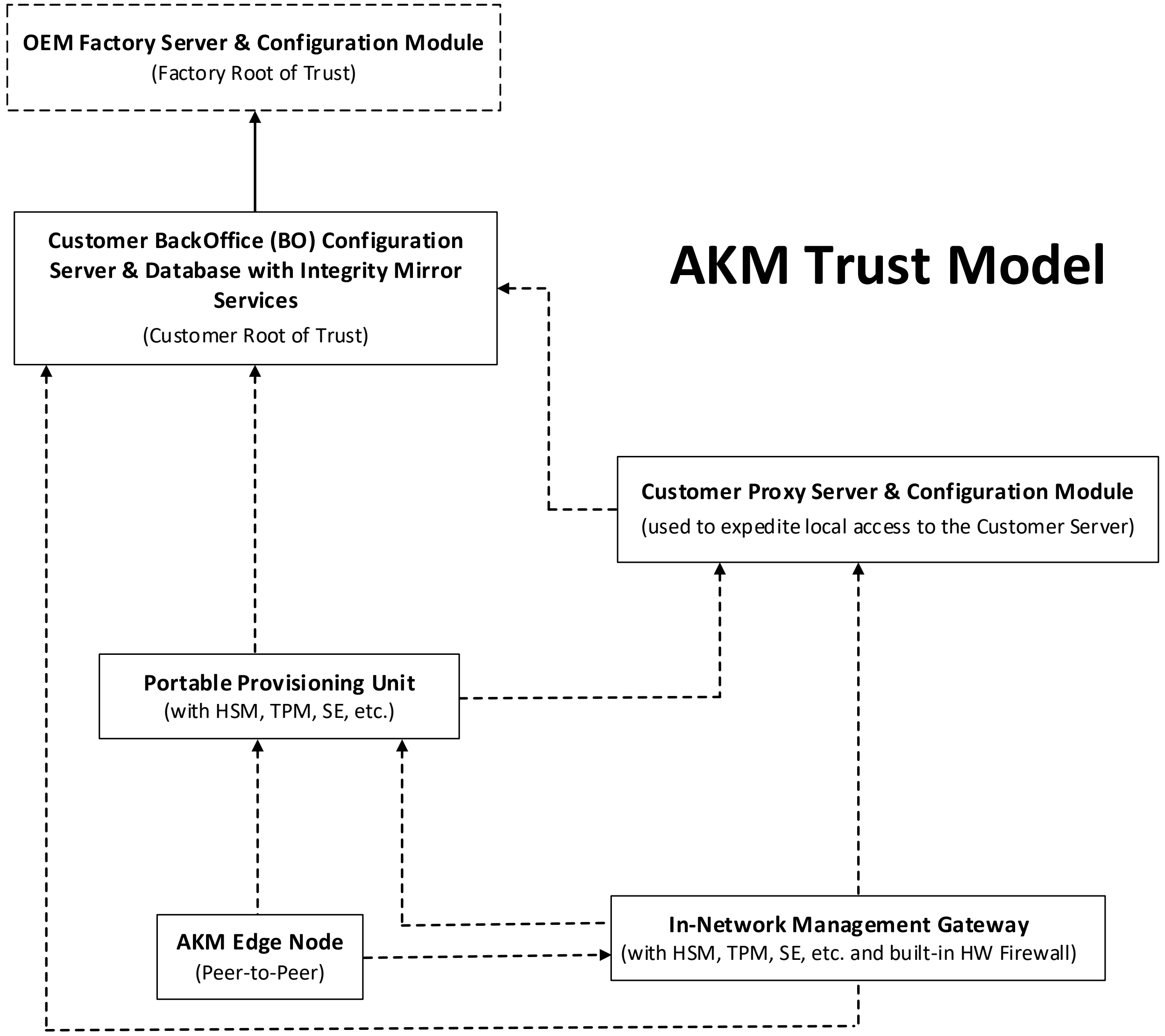
**Customer Proxy Server** is an optional element within the chain-of-trust and is usually required for customers whose operational domain spans across different geographical areas.

**Portable Configuration Module** is an optional element within the chain-of-trust, but in all probability, most implementations will utilize them. They are used primarily by maintenance and installation personnel for configuring units in the field as opposed to doing so either in the factory or remotely.+

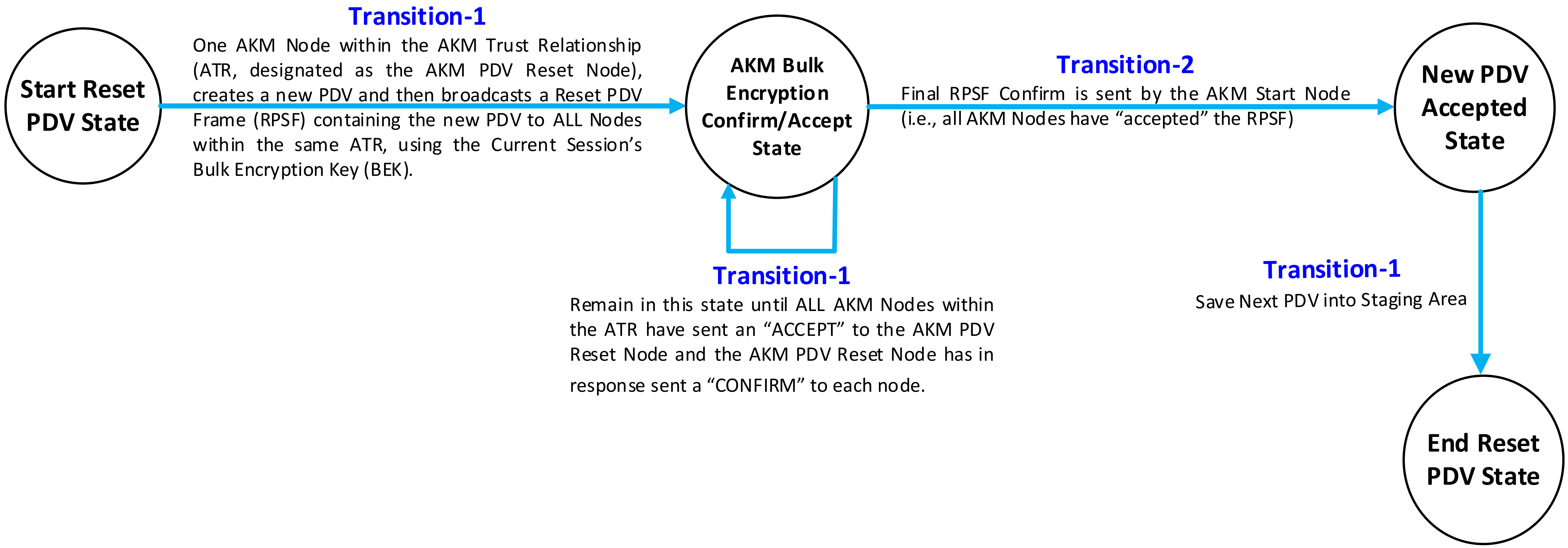
**In- Network Management Gateway** is again, an optional element within the chain-of-trust. It is typically used within a local network so that it can automatically manage the network locally without any necessity to be in constant contact with the BackOffice Server.

**AKM Edge Node** – This is a normal node within the AKM infrastructure and usually a primary element of the customer's overall application

## AKM Trust Model


OEM Factory Server & Configuration Module
(Factory Root of Trust)
Customer BackOffice (BO) Configuration Server & Database with Integrity Mirror Services
(Customer Root of Trust)
Customer Proxy Server & Configuration Module
(used to expedite local access to the Customer Server)
Portable Provisioning Unit
(with HSM, TPM, SE, etc.)
AKM Edge Node
(Peer-to-Peer)
In-Network Management Gateway
(with HSM, TPM, SE, etc. and built-in HW Firewall)

# Refreshing the PDV


Start Reset PDV State
Transition-1
One AKM Node within the AKM Trust Relationship (ATR, designated as the AKM PDV Reset Node), creates a new PDV and then broadcasts a Reset PDV Frame (RPSF) containing the new PDV to ALL Nodes within the same ATR, using the Current Session's Bulk Encryption Key (BEK).
AKM Bulk Encryption Confirm/Accept State
Transition-2
Final RPSF Confirm is sent by the AKM Start Node (i.e., all AKM Nodes have "accepted" the RPSF)
New PDV Accepted State
Transition-1
Remain in this state until ALL AKM Nodes within the ATR have sent an "ACCEPT" to the AKM PDV Reset Node and the AKM PDV Reset Node has in response sent a "CONFIRM" to each node.
Transition-1
Save Next PDV into Staging Area
End Reset PDV State

# Questions & Future Topics

This final section either answers or acknowledges obvious questions you might have regarding AKM. However, the explanation for many of the questions will require a deeper dive, perhaps even a presentation solely about the topic in question:

1) Where are the security credentials stored and what are the risks associated with where they are stored?

   *Response: This is an implementation specific question. The usual two answers are the credentials are either stored in normal, non-volatile memory or the credentials are stored within a tamper resistant secure hardware storage device with processing capability. The latter method is obviously the preferred one.*

2) How is synchronization maintained?

   *Response: The simplistic answer to this is provided in slide 25, where it describes the background thread of calculating next session credentials. The short answer is that the same algorithms are applied by the all of the nodes within a specific AKM Trust Relationship (ATR) on the same set of data, thus, yielding the same result for security credentials.*

3) Can an attacker instigate a resynchronization? If so, what is the mitigation, if any?

   *Response: The short answer to this question is maybe, but it is limited by an implementation configuration value. The default and very simplistic behavior is to issue a resynch after receiving a configured number of garbled frames. Once the relationship has resynched for the configured maximum number of times within a configured time period, garbled frames will simply be dropped for a second configurable amount of time, at which point, the default behavior will reset again. There are several variations for how resynch can be handled and will be described within its own forthcoming presentation specifically on this topic.*

## Questions & Future Topics

4) Can nodes join dynamically?  If so, is this an opportunity for an attack?

*Response There is no opportunity in AKM for nodes to truly join dynamically; thus, this is a non-issue with regard to a potential attack vector.  However, what can happen is that nodes can be part of a AKM Trust Relationship (ATR) and drop-out and rejoin again later via the optional arbitration session set of credentials.  To use the optional arbitration session credentials, all nodes that can be part of an ATR are known ahead of time (i.e. "registered" as part of the ATR).  However, not all nodes are active.  Thus, whenever an inactive node wishes to become active again, it transmits its request to reactivate via the arbitration ATR. The current designated arbiter node then engages with the formerly inactive node to integrate it back as an active node.  The full description of how this process works is beyond the scope of this primer and will be described fully in a forthcoming presentation specifically on this topic.*