Securing Application Layer Protocol CoAP using EDHOC and OSCORE under Contiki-NG OS

Dr. Radwan Dandah*
Dr. Ahmad Mahmoud Ahmad **
Rasha Sameer Ghadeer ***

(Received 20 / 2 / 2022. Accepted 22 / 5 / 2022)

□ ABSTRACT □

Securing the connection in IoT devices is an important research focus, Constrained Applications Protocol (CoAP) has received high attention as many IoT applications depend on it, but it suffers from some security vulnerabilities. Several protocols have been developed to protect CoAP, the most famous one is DTLS, which unfortunately imposes a great burden due to the limitations imposed on the capabilities of IoT devices. To overcome this problem, an alternative to DTLS protocol has recently been proposed. which consists of two novel protocols working side by side, the first one is Object Security for Constrained RESTful Environments (OSCORE) which provides authenticated encryption for the payload data, and the second is Ephemeral Diffie-Hellman Over COSE (EDHOC) which provides the symmetric session keys required for OSCORE. These protocols are relatively new and have few implementations.

This paper presents an implementation of EDHOC with OSCORE for CoAP protection under Contiki-NG OS, which is widely used for IoT restricted devices. The performance evaluation of the implementation is depicted on the Cooja simulator by comparing it with CoAP without protection from one hand and the use of DTLS from the other hand. The evaluation and comparison are effected in terms of latency and throughput. The results showed that the current implementation achieves key exchange for OSCORE protocol, and achieves the required protection for CoAP protocol, and gives lower latency and higher throughput compared to using DTLS.

Keywords: Internet of Things, Constrained Application Protocol (CoAP), Security, Datagram Transport Layer Security (DTLS), Object Security for Constrained RESTful Environments (OSCORE)

***Professor, Department of Computer Systems and Networks, Faculty of Informatics Engineering, Tishreen University, Lattakia, Syria. E-mail: radwan.dandah@tishreen.edu.sy**
****Associate Professor, Department of Computer Systems and Networks, Faculty of Informatics Engineering, Tishreen University, Lattakia, Syria. E-mail: ahmad.m.ahmad@tishreen.edu.sy**
*****PhD Student, Department of Computer Systems and Networks, Faculty of Informatics Engineering, Tishreen University, Lattakia, Syria. E-mail: r.ghadeer@tishreen.edu.sy**

حماية بروتوكول طبقة التطبيقات CoAP باستخدام البروتوكولين EDHOC و OSCORE في نظام التشغيل Contiki-NG

د. رضوان دنده*

د. أحمد محمود أحمد **

رشا سمير غدير ***

(تاريخ الإيداع 20 / 2 / 2022. قُبِل للنشر في 22/ 5 / 2022)

ملخّص

يعتبر تأمين الاتصال في أجهزة إنترنت الأشياء من المحاور البحثية المهمة، حظي بروتوكول التطبيقات المقيدة (CoAP) على اهتمام عالٍ حيث تعتمد العديد من تطبيقات إنترنت الأشياء عليه، ولكنه يعاني من بعض الثغرات الأمنية. من هنا تم تطوير العديد من البروتوكولات لحماية بروتوكول CoAP أشهرها بروتوكول DTLS، لكن العبء الذي يفرضه هذا البروتوكول كبير نسبة للقيود المفروضة على امكانيات أجهزة إنترنت الأشياء. للتغلب على هذه المشكلة، تم حديثاً اقتراح بديل لبروتوكول DTLS. يتكون هذا البديل من بروتوكولين يعملان جنباً إلى جنب، الأول هو أمان الكائن للبيئات المقيدة (OSCORE) والذي يوفر تشفيرًا مصدقًا لبيانات الحمولة و الثاني Ephemeral Diffie-Hellman Over COSE(EDHOC) والذي يوفر مفاتيح الجلسة المطلوبة لـ OSCORE. هذه البروتوكولات جديدة نسبياً وتوجد تطبيقات قليلة عليها.

يقدم هذا البحث تحقيق لبروتوكول EDHOC مع بروتوكول OSCORE لحماية بروتوكول CoAP على نظام تشغيل Contiki-NG المستخدم بكثرة للأجهزة المقيدة في إنترنت الأشياء. ومن ثم تقييم الأداء على محاكي Cooja من خلال المقارنة مع استخدام بروتوكول CoAP من دون حماية من جهة واستخدام بروتوكول DTLS كحماية من جهة أخرى. تمت دراسة السيناريوهات من حيث التأخير والإنتاجية، حيث أظهرت النتائج أن التحقيق الحالي يحقق تبادل المفاتيح لبروتوكول OSCORE كما أنه يحقق الحماية المطلوبة لبروتوكول CoAP، ويعطي تأخير أقل و إنتاجية أعلى مقارنة مع الحل الذي يعتمد على استخدام DTLS.

الكلمات المفتاحية: إنترنت الأشياء، بروتوكول التطبيقات المقيدة، حماية، أمن رزم بيانات طبقة النقل، أمان الكائن للبيئات المقيدة.

---

* أستاذ، قسم النظم والشبكات الحاسوبية، كلية الهندسة المعلوماتية، جامعة تشرين، اللاذقية، سورية.
E-mail: radwan.dandah@tishreen.edu.sy

** أستاذ مساعد، قسم النظم والشبكات الحاسوبية، كلية الهندسة المعلوماتية، جامعة تشرين، اللاذقية، سورية.
E-mail: ahmad.m.ahmad@tishreen.edu.sy

*** طالبة دراسات عليا (دكتوراه)، قسم النظم والشبكات الحاسوبية، كلية الهندسة المعلوماتية، جامعة تشرين، اللاذقية، سورية.
E-mail: r.ghadeer@tishreen.edu.sy

مقدمة:

يشير إنترنت الأشياء (IoT) إلى شبكة من الأشياء المقيدة والتي يمكن أن تتواصل مع بعضها البعض عبر الانترنت. والذي أدى إلى العديد من سيناريوهات التطبيق الجديدة، على سبيل المثال المباني الذكية وأتمتة المنشآت والمنازل وشبكات الكهرباء الذكية والنقل الذكي. العديد من أجهزة إنترنت الأشياء، والتي تسمى العقد، هي وحدات ذات موارد محدودة مثل الذاكرة وقوة المعالجة والطاقة (إذا كانت تعمل بالبطارية). يؤدي وجود موارد مقيدة إلى اتصالات شبكة مقيدة بسبب فقدان القنوات وعرض النطاق الترددي المحدود. من أجل التعامل مع هذا، تميل العقد المقيدة بالموارد إلى اعتماد نموذج اتصال غير متزامن ومتقطع، أي أنها تتعامل مع حركة مرور الشبكة وفقًا للفترات الزمنية للإرسال/الاستلام. لتوفير الطاقة، يمكن أن تصبح العقد غير متصلة بالإنترنت (السكون)، بين فترتين زمنيتين نشطتين، مما يؤدي إلى إطالة عمر البطارية بشكل كبير. لإدارة هذه القيود، يتم استخدام الوكلاء (proxy) كوسطاء لتمكين الوصول إلى عقد المخدم الغير متصلة بشكل دائم، عن طريق إعادة توجيه الطلبات من عقد الزبون وتخزين الاستجابات المرتبطة مؤقتاً ريثما يعود المخدم إلى حالة النشاط [1].

تم تطوير بروتوكول التطبيقات المقيدة CoAP والذي يعتبر من أهم بروتوكولات طبقة التطبيقات في إنترنت الأشياء مع دعم استخدام الوكلاء. تتطلب معظم التطبيقات اتصالات آمنة بين عقد الزبون والمخدم. ولهذه الغاية، يعتبر استخدام بروتوكول أمن رزم بيانات طبقة النقل DTLS طريقة لتحقيق اتصال آمن لـ CoAP. ينشئ بروتوكول DTLS قناة آمنة في طبقة النقل عبر بروتوكولات مخطط بيانات غير موثوق بها مثل بروتوكول UDP، ويوفر بذلك أماناً من خلال حماية رسائل بروتوكول CoAP بالكامل. وبما أن الوكلاء غير قادرين على قراءة رسائل CoAP المشفرة باستخدام بروتوكول DTLS، يجب أن تنتهي قناة DTLS عند الوكيل، بحيث يمكن للوكيل قراءة البيانات اللازمة من رسالة CoAP لتحقيق خدمات الوكيل. وبالتالي لايمكن انشاء قناة DTLS واحدة مباشرة بين الزبون والمخدم. بدلاً من ذلك، يجب إنشاء أول قناة آمنة بين الزبون والوكيل، ومن ثم انشاء قناة آمنة ثانية بين الوكيل والمخدم، وهذا بدوره يؤدي إلى القيدين التاليين: أولاً من الضروري أن يقوم الوكيل بإجراء معالجة أمنية مزدوجة لرسائل CoAP، حيث يتوجب عليه فك تشفير الرسالة المستلمة من الزبون على قناة DTLS، ومن ثم إعادة تشفير نفس الرسالة لتسليمها على قناة DTLS للمخدم، مما يؤثر على الأداء بشكل كبير. ثانياً، يلزم أن يكون الوكيل موثوقاً به، لأنه قادر على الوصول الكامل إلى محتوى حمولة رسائل CoAP. من الواضح أن فرض مثل هذا الحد من الثقة في الوكلاء ومقدمي الخدمات الذين يقومون بتشغيلهم يؤدي إلى كشف غير ضروري ومفرط للبيانات، والذي بدوره يخلق فرصاً للتلاعب بها ويثير بسهولة تداعيات الخصوصية [1].

للتغلب على هذه المشكلات بكفاءة، تم في عام 2019 اطلاق بروتوكول أمان الكائن للبيئات المقيدة (OSCORE) من قبل IETF [2]. يتخذ بروتوكول OSCORE نهج طبقة التطبيقات لحماية الرسائل استناداً إلى أمان الكائن، ويحمي بشكل انتقائي أجزاء معينة من رسالة بروتوكول CoAP في طبقة التطبيقات. وبالتالي يوفر اتصال آمن من طرف إلى طرف بين عقد الزبون والمخدم. يمكن بشكل خاص تشفير بعض الأجزاء من رسالة CoAP بينما يمكن المصادقة على الأجزاء الأخرى وحماية تكاملها. تمكن هذه الطريقة استخدام الوكلاء غير الموثوق بهم، والتي لاتزال قادرة على أداء المهام المقصودة دون الاطلاع على محتويات حمولة رسالة CoAP، وهذا يؤدي بدوره إلى تخفيف تهديدات الخصوصية التي يمكن أن يستغلها الوكلاء، وبالتالي الحفاظ على المجال الشخصي للمستخدمين البشريين المتعلق بالمعلومات المتبادلة والعمليات التي تنفذها نقاط النهاية المتصلة [3].

يعتمد بروتوكول OSCORE في تحقيقه على مفاتيح مولدة مسبقاً pre-shared key، يمكن استكمال OSCORE ببروتوكول EDHOC [4]، وهو بروتوكول تبادل مفاتيح غير متماثل يستخدم تنسيق رسالة COSE لاشتقاق مفاتيح التشفير التي يمكن استخدامها بعد ذلك لتشفير رسائل OSCORE بشكل متماثل. بالإضافة إلى ذلك، من أجل أن يكون كلا البروتوكولين مناسبين تمامًا للأجهزة والشبكات المقيدة، فإنهما يستفيدان من تنسيقات التشفير المنخفضة التمثيل الثنائي الموجز [5] (CBOR)، وتوقيع وتشفير كائن CBOR (COSE) [6].

الدراسات المرجعية:

يوجد الكثير من الدراسات المرجعية التي اهتمت بموضوع الحماية لبروتوكول التطبيقات المقيدة. أكثر الدراسات كانت تعتمد على بروتوكول DTLS وتقييم أداءه والعبء الذي يفرضه وخاصة أن حجم الرسائل الصادرة عنه كبير نسبياً وغير مناسب في حال السيناريوهات التي تكون فيها الأشياء غير موصولة بشكل دائم بالطاقة. قدم البحث [7] مقترحاً لحماية البروتوكول CoAP باستخدام بروتوكول أمن رزم بيانات طبقة النقل DTLS في نظام IoT مصمم لتمييز الوجوه مبني باستخدام المكتبة OpenCV المقدمة من قبل السحابة. قام البحث في البداية بتحليل البنية المقترحة من الناحية الأمنية، ليتم بعدها دراسة عبء استخدام البروتوكول DTLS، حيث تم تحقيق ذلك من خلال المقارنة بين نقل الصور بوجود حماية وبعدمها، ليتبين لنا أن استخدام بروتوكول DTLS يفرض عبء على الأشياء، حيث يزيد استهلاك البطارية بنسبة 12% كما أنه يزيد زمن استخدام المعالج بمقدار الضعف تقريباً.

بعد أن قامت IETF باقتراح بروتوكول OSCORE توجهت العديد من الجهود البحثية نحو كتابة تحقيقات للبروتوكول وفق لغات مختلفة لتتناسب مع تطبيقات إنترنت الأشياء وتقييم أداءه ومقارنته ببروتوكول DTLS. قدمت الورقة البحثية [8] في عام 2020 مقارنة بين ثلاثة بروتوكولات حماية تستخدم لبروتوكول CoAP وهي DTLS, TLS, OSCORE وتقييم الأداء نسبة لاستهلاك الطاقة والإنتاجية والتأخير. أوجدت النتائج أن بروتوكول OSCORE يتفوق على باقي البروتوكولات السابقة، حيث أنه يقدم أقل تأخير وأكثر إنتاجية وأقل استهلاك لطاقة البطارية. تم التركيز في هذا البحث على تحقيق لبروتوكول OSCORE وبروتوكول CoAP بلغة الجافا والذي يدعى Californium. كما أنه قد وضح إلى ضرورة استخدام بروتوكول EDHOC مع بروتوكول OSCORE وإعادة المقارنات في الأعمال المستقبلية.

تم في [3] ، 2021 أول تطبيق مفتوح المصدر ومتوافق مع المعايير لبروتوكول OSCORE لنظام تشغيل Contiki-NG وعلى نطاق أوسع لأجهزة إنترنت الأشياء المقيدة من دون وجود أي بروتوكول لتبادل المفاتيح، وإنما بالاعتماد على مفاتيح مولدة مسبقاً. تم مقارنة أداء OSCORE مقابل كل من سيناريو النقل غير الآمن باستخدام بروتوكول CoAP العادي، وسيناريو بديل آمن باستخدام بروتوكول CoAP عبر DTLS، حيث أجريت المقارنات على سيناريو يحتوي على وكيل لتبيان تفوق استخدام بروتوكول OSCORE على بروتوكول DTLS في حال وجود وكيل في سيناريو انترنت الاشياء، تمت المقارنات بعد تجاهل مصافحة DTLS وتبادل مفاتيح OSCORE وذلك بسبب عدم توفر تحقيق لتأسيس مفاتيح الأمان لبروتوكول OSCORE على Contiki-NG. قدمت هذه الدراسة البحثية أول تقييم تجريبي للأداء لإصدار متوافق مع معايير OSCORE على جهاز حقيقي لإنترنت الأشياء. اعتمد التقييم على التطبيق المتوافق مع المعايير الخاصة بـ OSCORE لنظام التشغيل Contiki-NG، ولا سيما المنصة المحدودة الموارد Zolertia Firefly المجهزة بنظام CC2538. أظهرت النتائج التجريبية أن OSCORE يعرض أداءً أفضل بشكل معتدل من DTLS في مقاييس مهمة، أي حمل الإرسال اللاسلكي، ووقت الرحلة، واستخدام الذاكرة بالإضافة إلى كفاءة الطاقة للخوادم المقيدة، مع استمرار السماح للوكيل بأداء العمليات التي يجب أن يقوم بها.

أما في الأبحاث [9] و[10] و [11]، فقد تم تحقيق بروتوكول OSCORE مع بروتوكول EDHOC بلغات برمجية و أنظمة مختلفة، ففي الورقة البحثية [9] قدم الباحثون أول تطبيق لبروتوكلي EDHOC و OSCORE باستخدام لغة برمجة Rust، كما أظهروا قابلية التطبيق في سيناريو لزبون حقيقي ومخدم للموارد على أجهزة STM32 المقيدة. أما في الورقة البحثية [10]، فقد قاموا بتطبيق البروتوكولين اللذان يتيحان طريقة جديدة للاتصال الآمن المصمم للأجهزة المدمجة على GitLab. تم كتابة واختبار المكتبات الجديدة، التي تنفذ البروتوكولات المذكورة، بمساعدة خط أنابيب التكامل المستمر (Continuous Integration). ثم جمع بعض مقاييس الأداء وتشغيل أمثلة على الصفحات بنجاح على جهاز كمبيوتر قياسي بالإضافة إلى جهاز مضمن محدد. قدم البحث [11] تصميم أربع مكتبات للبرامج الثابتة لبروتوكولي OSCORE و EDHOC تستهدف بشكل خاص المتحكمات الدقيقة وتقييمها التفصيلي. بتعبير أدق، تصميم مكتبات µOSCORE و µEDHOC لوحدات التحكم الدقيقة العادية ومكتبات µOSCORE-TEE و µEDHOCTEE لوحدات التحكم الدقيقة مع بيئة تنفيذ موثوقة (TEE)، مثل المتحكمات الدقيقة التي تتميز بـ ARM TrustZone-M. يتعلق تصميم البرامج التي يوفرها هذا البحث بحقيقة أن المهاجمين قد يستغلون ثغرات البرامج الشائعة، على سبيل المثال، فيضان المخزن المؤقت (buffer overflows) في البروتوكول أو نظام التشغيل أو التطبيق لخرق أمان البروتوكول. تحقق µOSCORETEE و µEDHOC-TEE فصل عمليات التشفير والمفاتيح عن باقي البرامج الثابتة، والتي قد تكون عرضة للخطر. قدم البحث أيضاً تقييمًا لعمليات التحقيق السابقة من حيث متطلبات RAM / FLASH وسرعة التنفيذ والطاقة على مجموعة واسعة من وحدات التحكم الدقيقة.
من الدراسات المرجعية نلاحظ أهمية وجود تحقيق لبروتوكول EDHOC واستخدامه لتبادل مفاتيح مع بروتوكول OSCORE في معظم أنظمة التشغيل التي يتم استخدامها في تطبيقات إنترنت الأشياء، ولاسيما نظام Contiki-NG [12] المستخدم في هذا البحث. Contiki-NG نظام تشغيل مطور من Contiki مفتوح المصدر يستخدم لأنظمة شبكية مقيدة بالذاكرة مع التركيز على الأجهزة اللاسلكية منخفضة الطاقة. يدعم اتصالات IP v6 (موثوقة وآمنة) القائمة على المعايير، كما يركز على منصات انترنت الأشياء الحديثة مثل ARM Cortex M3 وغيرها الكثير ويستخدم بكثرة في الدراسات البحثية والعملية كونه يدعم العديد من البروتوكولات وأجهزة إنترنت الأشياء.

أهمية البحث وأهدافه:

يمثل انتشار أجهزة إنترنت الأشياء تحدياً أمنياً كبيراً، والتي تعد أهدافاً رئيسية للمهاجمين. وبما أن العديد من هذه الأجهزة سيتم استخدامها لفترة أطول بكثير من الأجهزة الإلكترونية التقليدية، وبالتالي يمكن أن يكون استخدام البروتوكولات القديمة غير المشفرة للاتصال بمثابة ناقل للهجوم أيضًا، مما يترك حركة المرور مفتوحة للفحص والتلاعب، وهذا هو المجال الذي ركزنا عليه، من خلال الاعتماد على بروتوكول جديد OSCORE تم اطلاقه في شهر تموز لعام 2019، مصمم لحماية بروتوكول CoAP ذو عبء خفيف، والذي يستخدم بشكل شائع للأجهزة المقيدة في شبكات إنترنت الأشياء. وإضافة بروتوكول آخر جديد يدعى EDHOC، الغرض منه هو استكمال OSCORE وتوسيعه للسماح بتبادل آمن للمفاتيح. التحقيقات الحالية لهذين البروتوكولين قليلة جداً. تأتي أهمية هذا البحث في تحقيق لبروتوكول OSCORE مع بروتوكول EDHOC على نظام تشغيل Contiki-NG المستخدم بكثرة في تطبيقات إنترنت الأشياء. من هنا يهدف هذا البحث إلى شرح البروتوكولين السابقين وتحقيقهما معاً بالإضافة إلى القيام

بتقييم أداء التحقيق بمقارنته بسيناريو غير آمن وسيناريو آمن باستخدام بروتوكول DTLS بالاعتماد على عدة بارامترات أهمها التأخير والإنتاجية. تم في هذا البحث أول مقارنة لاستخدام بروتوكول تبادل مفاتيح مع بروتوكول حماية (EDHOC+OSCORE) من جهة مع بروتوكول (DTLS) مع مصافحة من جهة أخرى في نظام تشغيل Contiki-NG، حيث نلاحظ أنه في الدراسات المرجعية يتم إهمال المصافحة في كلا البروتوكولين.

طرائق البحث ومواده:

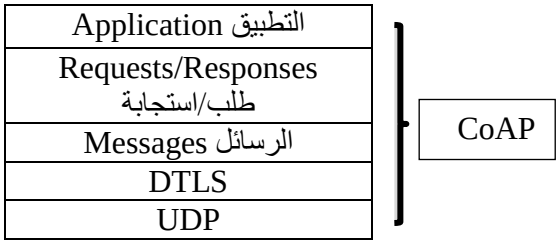
يعتمد البحث الطرق الوصفية والتطبيقية والتجريبية، حيث يبدأ البحث بعرض لبروتوكول CoAP يليه شرح مختصر لبروتوكولات DTLS و OSCORE و EDHOC و CBOR و COSE مع توضيح بنية عمل كل منها، ومن ثم يتم شرح البنية العملية المنجزة. نتبع بعد ذلك الدراسة التطبيقية لشرح التطبيق المقترح، وننهي البحث بإجراء تجارب لتقييم أثر استخدام البروتوكولين EDHOC و OSCORE مع البروتوكول CoAP من ناحية الإنتاجية والتأخير.

1- بروتوكول التطبيقات المقيدة (Constrained Application Protocol):

بروتوكول CoAP عبارة عن بروتوكول ويب تم تصميمه لأنظمة الزمن الحقيقي المبنية على استخدام أجهزة IoT صغيرة الحجم، ذات التكلفة قليلة والمحدودة الموارد. يمكن لهذا البروتوكول من ارسال بيانات الأشياء بالإضافة إلى رسائل التحكم، كما يؤمن عمليات تبادل للرسائل غير المتزامنة باستخدام بروتوكول رزم بيانات المستخدم UDP، يعتمد CoAP على نموذج مخدم/زبون مثل بروتوكول HTTP، ويمثل عملية الارسال والاستقبال بنفس الطريقة. يتم تعريف مجموعة من الموارد عن طريق ربطها بمعرف يسمى معرف الموارد المحدد Uniform Resource Identifiers (URIs) باستخدام طرائق النقل مثل GET، POST ، PUT، DELETE أي (إحضار – إرسال – تعديل – حذف) مورد من مخدم، كما أنه يتم الرد بإحدى أرقام الاستجابة والذي يعبر عن الاستجابة الصحيحة أو الخاطئة. على عكس بروتوكول HTTP، يقوم بروتوكول CoAP بتبادل الرسائل بطريقة غير مباشرة عبر UDP، لذلك ينبغي عليه تحقيق كل الخدمات المقدمة من قبل TCP، لاسيما الأمنية منها [13].

2- بروتوكول أمن رزم بيانات طبقة النقل (Datagram Transport Layer Security DTLS) :

تم اقتراح بروتوكول DTLS 1.2 من قبل IETF بالمستند رقم (RFC 6347)، وهو يستخدم لتأمين مسار الشبكة، ويشبه DTLS بروتوكول أمن طبقة النقل (Transport Layer Security) TLS مع تعديلات معينة لنستطيع استخدامها في البيئات التي تتعامل مع بروتوكولات نقل غير موثوقة مثل UDP، من هذه التعديلات: معالجة فقدان الرسالة، إعادة ترتيب الرسالة، تعديل حجم الرسالة [14]. يمكن استخدام بروتوكول DTLS لتأمين بروتوكول CoAP، حيث يضمن تحقيق الأمن الشامل للتطبيقات المختلفة عن طريق العمل بين طبقة التطبيقات وطبقة النقل كما في الشكل (1).

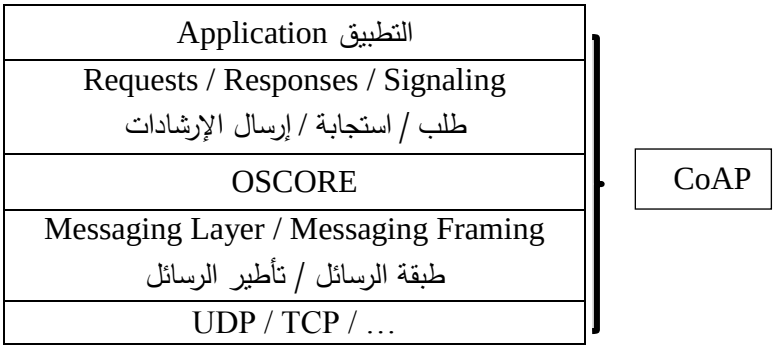


الشكل (1) : طبقات بروتوكول CoAP المحمي ببروتوكول DTLS

3- بروتوكول أمان الكائن للبيئات المقيدة Object Security for Constrained RESTful Environments (OSCORE) :

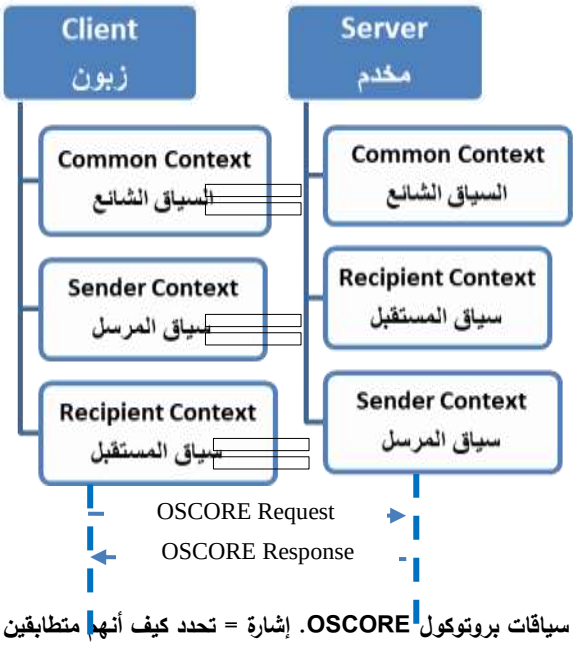
تم اقتراح بروتوكول OSCORE من قبل IETF بالمستند رقم (RFC 8613)، هو بروتوكول يعمل على طبقة التطبيقات يوفر حماية لبروتوكولCoAP من خلال توفير التشفير المصدق طرف إلى طرف وحماية إعادة التشغيل وربط الطلبات والاستجابات. في الوقت نفسه، يسمح OSCORE بعمليات الوكيل. بتعبير أدق، يحمي OSCORE: 1) رمز الطريقة/الاستجابة، 2) URI للمورد المطلوب و 3) الحمولة. حزمة OSCORE لها نفس بنية حزمة CoAP. الاختلاف بين الاثنين هو أن حمولة حزمة OSCORE مشفرة ومحمية بسلامة باستخدام خوارزمية تشفير مصدق مع البيانات المرتبطة(AEAD) Authenticated Encryption with Associated Data . تستخدم خوارزمية AEAD مفاتيح متناظرة مشتركة بين الزبون والمخدم، والتي يمكن إما أن تكون مُسبقة الانشاء أو مُنشأة ببروتوكول إنشاء مفاتيح مثل بروتوكول EDHOC. يتم تحديد حزم OSCORE بواسطة حقل خيار OSCORE، والذي يمكن أن يحتوي أيضًا على بارامترات إضافية [2].

يمكن استخدام OSCORE لكل من النقل غير الموثوق به حيث تختلف هذه الطرق فقط في طبقة رسائل CoAP، وهي غير محمية بـ OSCORE. بالإضافة إلى ذلك، يحمي OSCORE التفاعلات RESTful مثل طريقة الطلب والمورد المطلوب وحمولة الرسالة كما هو موضح في الشكل (2). نظرًا لأن OSCORE تحمي فقط معلومات طبقة التطبيق ذات الصلة، فإن حمل الرسالة يكون ضئيلاً [2].



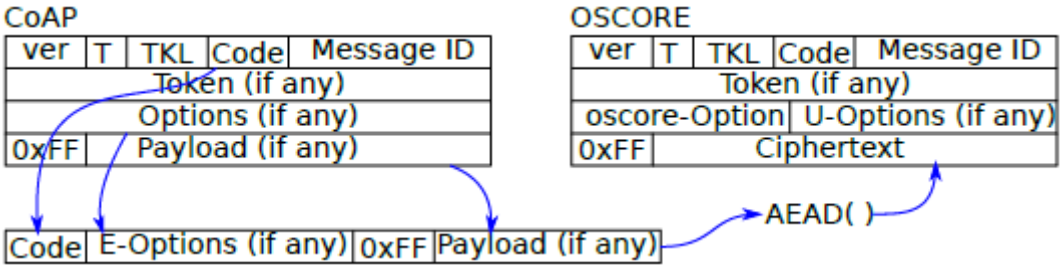
الشكل (2): طبقات CoAP + OSCORE

سياقات أمان (Security Context) لبروتوكول OSCORE: تحتفظ كل نقطة نهاية OSCORE بجميع المعلومات التي تتطلبها في ثلاثة سياقات أمنية كما يظهر في الشكل(3): سياق مشترك وسياق المرسل وسياق المستلم. السياق الشائع أو السياق المشترك، كما يشير الاسم، مشترك بين كل من الزبون والمخدم. وهو يتألف من معرفات لخوارزمية (AEAD)مثل خوارزميتي AES-CCM-16-64-128 و HMACSHA256، السر الرئيسي Master Secret وهو مفتاح مشترك وMaster Salt ومعرف السياق ID Context وCommon IV. يمتلك كل من زبون ومخدم OSCORE سياق المرسل والمستقبل. البارامترات معرف المرسل Sender ID ومفتاح المرسل Sender Key لسياقات المرسل متطابقة مع البارامترات معرف المستلم Recipient ID ومفتاح المستلمRecipient Key لسياقات المستلم. مفتاح المرسل Sender Key ومفتاح المستلم Recipient Key هما مفتاحان متماثلان مشتقان من السر الرئيسي. معرف المرسل Sender ID ومعرف المستلم Recipient ID هما معرفات تستخدم لتحديد سياقات المرسل/المستلم [11] .



الشكل(3): سياقات بروتوكول OSCORE. إشارة = تحدد كيف أنهم متطابقين بين الزبون والمخدم

التحويل بين CoAP و OSCORE والعكس بالعكس: يتم تحويل رسالة CoAP إلى رسالة OSCORE قبل إرسالها، والعكس صحيح عند استلامها. يوضح الشكل(4) تحويل رسالة CoAP إلى OSCORE. يشكل حقل الكود، وبعض الخيارات التي تحتاج إلى الحماية (تسمى خيارات E) والحمولة الاختيارية نصًا عاديًا، يتم تشفيره وحماية سلامته. نص التشفير الناتج هو حمولة حزمة OSCORE. رمز حزمة OSCORE ثابت – 0.02 (POST) للطلبات و 2.04 (Changed) للاستجابات. بالإضافة إلى ذلك، تحتوي حزمة OSCORE على حقل خيار OSCORE، والذي يستخدم لتمييز حزمة OSCORE عن حزمة CoAP. علاوة على ذلك، في بعض الحالات، ينقل خيار OSCORE المحددات المستخدمة لإنشاء وتحديد السياقات في الطرف المتلق [11].



الشكل (4) تحويل رسالة OSCORE إلى رسالة CoAP

4- تمثيل الكائن الثنائي الموجز Concise Binary Object Representation (CBOR) :

تم اقتراح CBOR من قبل IETF بالمستند رقم (RFC 7049)، وهو تمثيل ثنائي معياري للبيانات المهيكلة يتبع أهداف تصميم محددة لم يتم تناولها بشكل جيد بواسطة تنسيقات البيانات الحالية. تتضمن أهداف التصميم هذه تقليل حجم الرسالة بالإضافة إلى تقليل مقدار الكود المطلوب لمعالجة الرسائل. CBOR هي نسخة موسعة من نموذج

بيانات JSON لكن CBOR تحدد نموذج البيانات الخاص بها دون التخطيط للتوافق مع الإصدارات السابقة. يتم استخدامه بدلاً من JSON في البيئات المقيدة [5].

5- توقيع كائن CBOR وتشفيره (CBOR Object Signing and Encryption (COSE:

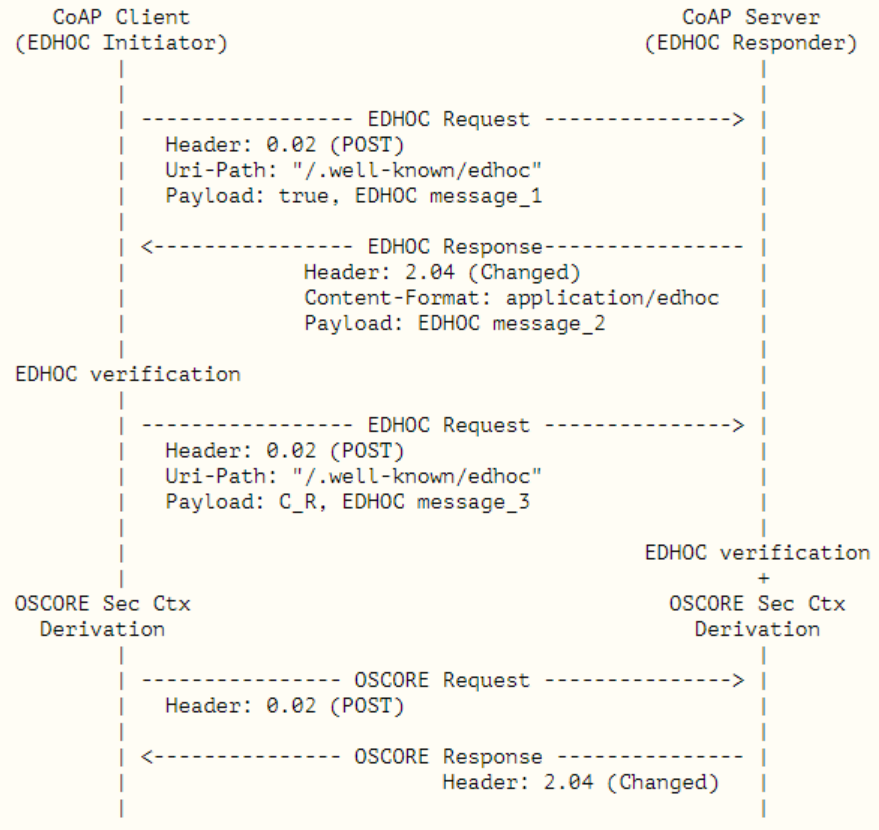
تم اقتراح COSE من قبل IETF بالمستند رقم (RFC 8152). يتم تعريف CBOR على أنه رمز صغير وتنسيق بيانات حجم الرسالة ولكن هناك حاجة إلى خدمات الأمن. يحدد بروتوكول COSE كيفية إنشاء ومعالجة التوقيعات وأكواد مصادقة الرسائل والتشفير باستخدام CBOR، بالإضافة إلى تمثيل مفاتيح التشفير في تسلسل [6] CBOR.

6- بروتوكول(EDHOC) Ephemeral Diffie-Hellman Over COSE:

EDHOC عبارة عن بروتوكول تبادل مفاتيح Diffie-Hellman صغير الحجم وخفيف الوزن وموثق بمفاتيح سريعة الزوال. يوفر EDHOC المصادقة المتبادلة وسرية التوجيه وحماية الهوية. EDHOC مخصص للاستخدام في سيناريوهات مقيدة، يستخدم تنسيق رسالة COSE لاشتقاق مجموعة من البارامترات ومفاتيح التشفير التي يمكن استخدامها بعد ذلك لتشفير رسائل OSCORE المتبادلة مابين الزبون والمخدم. يتم إنشاء سياق أمان OSCORE، من خلال إعادة استخدام COSE للتشفير، وCBOR للترميز، وCoAP للنقل، بحيث يمكن إبقاء حجم الكود الإضافي منخفضًا للغاية[4] .

يستخدم EDHOC بروتوكولًا من ثلاثة رسائل لإنشاء المفاتيح والتفاوض بشأن محددات الأمان. يُطلق على الأطراف التي تتبادل الرسائل اسم البادئ (Initiator) والمستجيب (Responder) اللذان ينشئان زوجًا جديدًا من مفاتيح ECDH سريع الزوال لكل جلسة، ويتبادلان الجزء العام من مفاتيح ECDH الخاصة بهما، ويحسبان السر المشترك Master Secret، ويستنبطان مفاتيح التطبيق المتماثلة. يمكن مصادقة الرسائل باستخدام مفاتيح عامة خام (RPK) يمكن أن تحتوي على مفاتيح توقيع أو مفاتيح ECDH ثابتة، بالإضافة إلى شهادات المفاتيح العامة.

يتكون EDHOC من ثلاث رسائل (MSG1 و MSG2 و MSG3)، بالإضافة إلى رسالة خطأ EDHOC (MSG ERR) التي يتم نقلها كحمولة في رسائل CoAP القابلة للتأكيد إلى مسار uri /.well-known/edhoc ، حيث يكون زبون CoAP هو بادئ EDHOC ويكون مخدم CoAP هو مستجيب EDHOC. يتم نقل MSG1 و MSG3 في طلبات POST، ويتم نقل MSG2 في استجابة 2.04 (متغيرة). عندما يكون حجم MSG أكبر من 64 B، يتم استخدام آلية النقل Block-Wise للتجزئة التي يوفرها بروتوكول CoAP.



الشكل (5) البروتوكولين EDHOC و OSCORE يعملان بالتتابع

يوضح الشكل (5) زبون ومخدم CoAP يقومان بتشغيل EDHOC كبادئ ومستجيب، على التوالي. بمعنى، يرسل الزبون طلب POST إلى مورد EDHOC محجوز على المخدم، افتراضيًا في مسار Uri "/.well-known/edhoc". تتكون حمولة الطلب من القيمة البسيطة (0xf5) "true" CBOR المتسلسلة مع رسالة EDHOC_1، والتي تتضمن أيضًا معرف اتصال EDHOC C_I الخاص بالزبون [15].

يؤدي هذا إلى تشغيل تبادل EDHOC على المخدم، والذي يرد باستجابة 2.04 (تم التغيير). تتكون حمولة الاستجابة من رسالة EDHOC_2، والتي تتضمن أيضًا معرف اتصال EDHOC C_R الخاص بالمخدم. أخيرًا، يرسل الزبون طلب POST إلى نفس مورد EDHOC المستخدم سابقًا لإرسال رسالة EDHOC_1. تتكون حمولة الطلب من معرف اتصال EDHOC C_R المتسلسل مع رسالة EDHOC_3. بعد حدوث هذا التبادل، وبعد عمليات التحقق الناجحة على النحو المحدد في بروتوكول EDHOC، يمكن للزبون والمخدم اشتقاق سياق أمان OSCORE، بعد ذلك يمكنهم استخدام OSCORE لحماية اتصالاتهم وفقًا لـ [RFC 8613] [15].

7- الجزء العملي:

في البداية لم يتم التمكن من تنزيل نظام التشغيل Contiki-NG المفتوح المصدر والمتوفر على github [15] كسلسلة أو كنظام مستقل على نظام تشغيل ابونتو، وذلك بسبب احتياجه إلى مجموعة من الحزم التي لم نستطع توفيرها. إنما تم تنزيل Contiki-NG كصورة docker image على نظام تشغيل ubuntu وتنزيل كافة الحزم اللازمة للعمل بعدة خطوات مشروحة في المرجع [17].

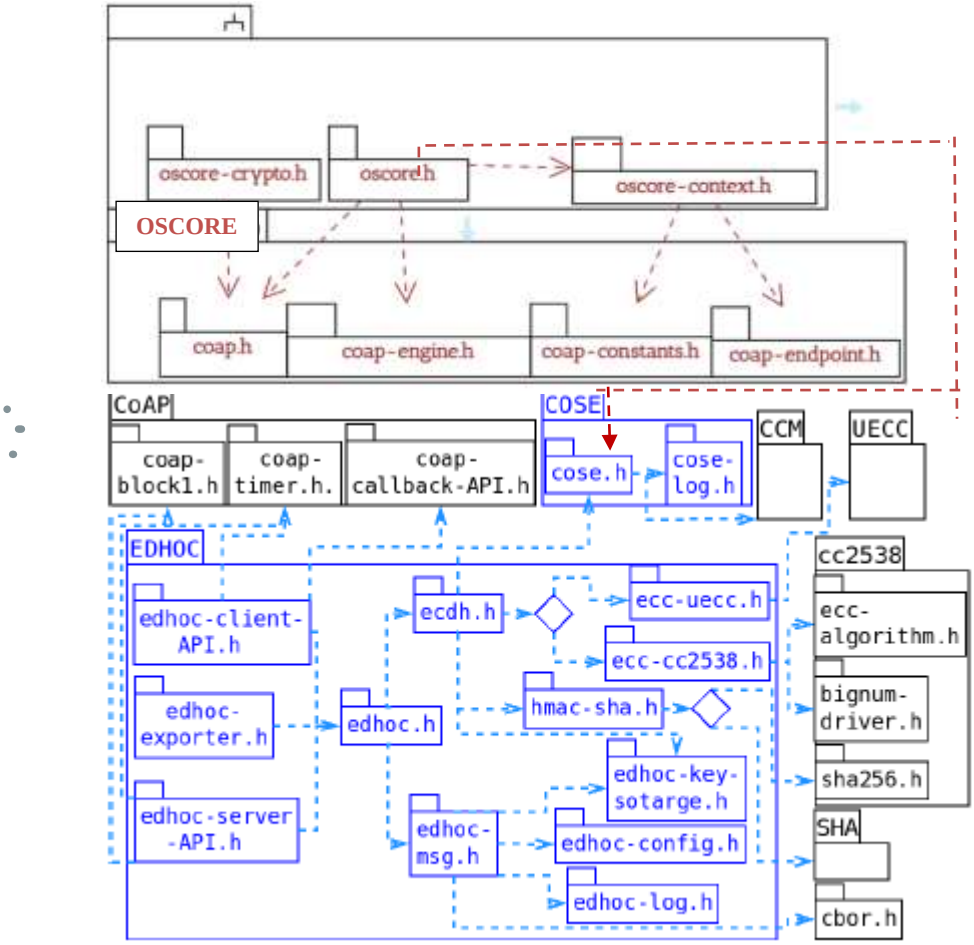
لبدء جلسة داخل حاوية docker يجب كتابة التعليمة التالية :

$ contiker

عندها نكون في المسار home/user/contiki-ng/ في الحاوية وموجودين بشكل مباشر بالنسخة المحلية الخاصة بنا لنظام تشغيل Contiki-NG.

قمنا بالاستفادة من نسخة OSCORE المقدمة في الدراسة البحثية [3]، والتي تضمنت تحقيق مفتوح المصدر لبروتوكول OSCORE موجود على github [18]. لايوجد في هذا التحقيق أي خوارزمية مستخدمة لتبادل المفاتيح بين الزبون والمخدم، حيث يتم وضع مفتاح السر الرئيسي Master Secret و المفتاح Master Salt بشكل يدوي في الكود، بالإضافة إلى معرف المرسل Sender ID ومعرف المستقبل Reciever ID من جهة الزبون التي توضع قيمها بشكل معاكس لمعرف المرسل ومعرف المستقبل من جهة المخدم. كما أنه يوجد تحقيق جزئي لبروتوكول EDHOC موجود علgithub بلغة C، ولكن هذا التحقيق غير معتمد من قبل Contiki-NG. تم الاستفادة من التحقيقين السابقين ودمجهم والتعديل عليهم حتى نشأ لدينا التحقيق الموضح في الشكل (6)، هذا التحقيق يتيح لنا إقامة سيناريوهات وتطبيقات لإنترنت الأشياء على نظام تشغيل Contiki-NG واستخدام بروتوكول EDHOC لإنشاء المفاتيح وتبادلهم مابين الزبون والمخدم للبدأ بإنشاء سياق أمان بروتوكول OSCORE، في هذا التحقيق يتم استخدام بروتوكول CoAP للنقل و COSE للتشفير و CBOR للترميز.

يتضمن مخطط حزمة UML الوارد في الشكل (6) تنفيذ وحدات EDHOC و COSE و OSCOREبالإضافة إلى الوحدات النمطية والمكتبات مفتوحة المصدر المعاد استخدامها. تتضمن وحدة COSE العمليات الضرورية التي تستخدمها EDHOC ، وهي بنية مشفرة COSE Encrypt تستخدم لعمليات التشفير/فك التشفير باستخدام خوارزمية AEAD. يتم تحقيق واجهة وحدة EDHOC من خلال أربع واجهات برمجة تطبيقات متميزة. يقوم-edhoc-clien API.hبتصدير الوظائف لاستدعاء بروتوكول EDHOC كمنشئ زبون CoAP ويصدر edhoc-server-API.h الوظائف لاستخدامها من عملية المخدم الرئيسية (المستجيب) ومن مورد مخدم.EDHOC . بالإضافة إلى ذلك، فإن أداة edhocexporter.h توفر الوظائف لتصدير سياق أمان التطبيق من المفتاح المشترك المتماثل النهائي المشتق إلى بروتوكول OSCORE، بينما يمكن إدارة مستودع تخزين المفاتيح باستخدام واجهة برمجة تطبيقات-edhoc-key storage.h.



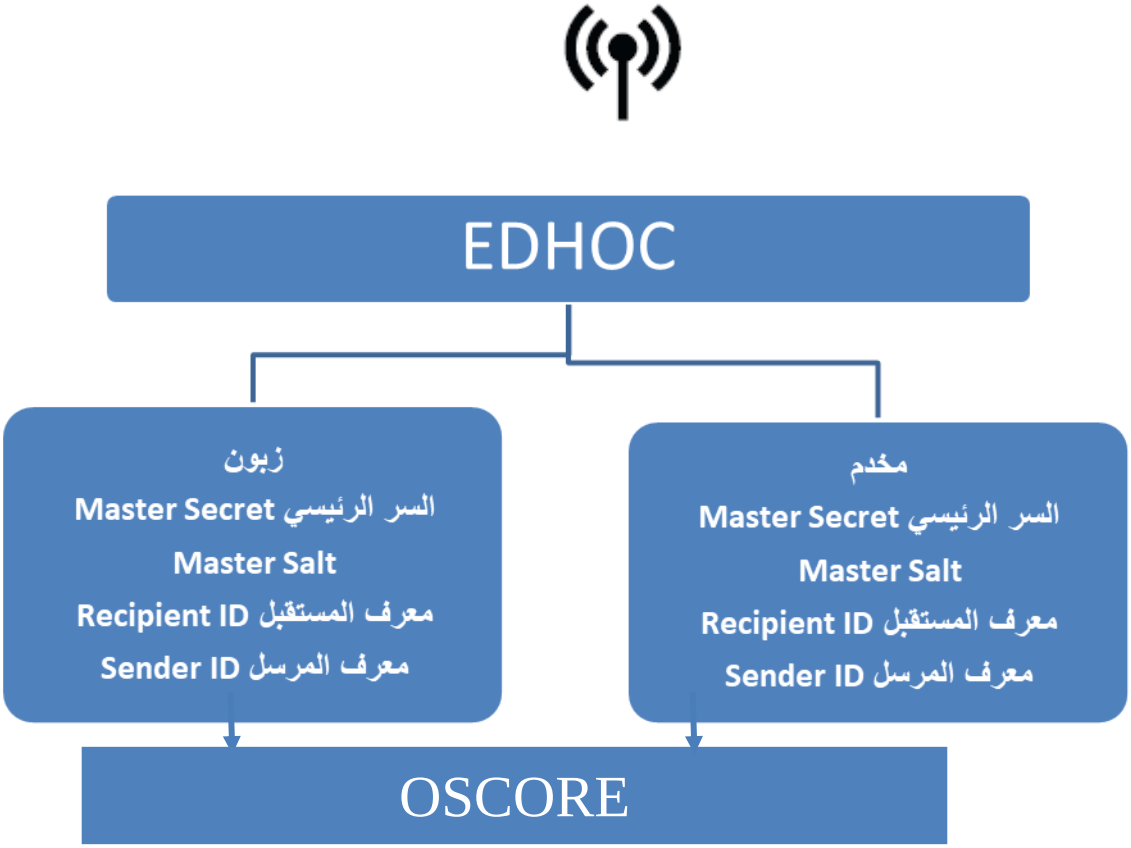
الشكل (6) مخطط حزمة UML للبروتوكولين EDHOC و OSCORE

يتم استخدام تطبيق Contiki-NG لـ CoAP لنقل رسائل EDHOC، ويتم استخدام تطبيق CCM Contiki-N G المستند إلى AES 128 لتشفير AEAD، وتنفيذ CBOR من مستودع مجموعة OSCORE للترميز. يمكن تكوين بروتوكول EDHOC لتشغيل عملية تشفير ECDH و SHA-2.

أما بالنسبة لبروتوكول OSCORE فهو يحتوي على ثلاثة مكتبات مهمة، المكتبة الأولى oscore-crypto.h والتي تقوم بتشفير وفك تشفير حزم CoAP، أما المكتبة oscore-context.h المختصة بتأسيس السياقات الثلاثة التي تحدثنا عليها في الفقرة (3) سابقاً، السياق الشائع وسياق المرسل و سياق المستقبل. تحمي نقطة النهاية الرسائل المرسلة باستخدام سياق المرسل، وتتأكد من الرسائل المستقبلة باستخدام سياق المستقبل. سياق المرسل وسياق المستقبل يتم اشتقاقها من السياق الشائع وبيانات أخرى. أما المكتبة oscore.h فهي المكتبة المسؤولة عن التحويل مابين رسالة OSCORE و رسالة CoAP.

كما في الشكل (7)، نستخدم EDHOC لإنشاء سر رئيسي Master Secret و Master Salt ومعرف المرسل ومعرف المستلم لكل من الزبون والمخدم. يتم مشاركة السر الرئيسي و Master Salt من قبل الطرفين، بينما يتم وضع معرف المرسل ومعرف المستلم بشكل معكوس مابين الزبون والمخدم. يتم استخدام هذه المحددات في بروتوكول

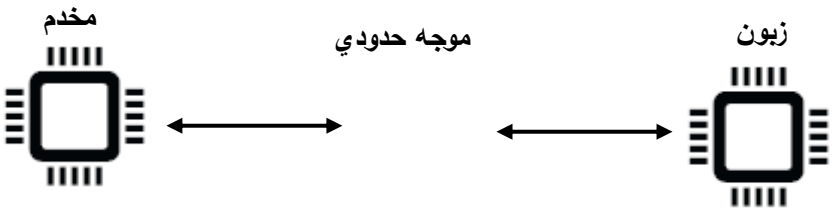
OSCORE لاشتقاق مفاتيح تشفير المحتوى ومفاتيح فك التشفير التي تُستخدم بعد ذلك لتشفير حركة مرور الرسائل بين الزبون والمخدم.



الشكل (7) تأسيس سياق أمان oscore بالاعتماد على بروتوكول EDHOC

النتائج والمناقشة:

لتقييم أداء التحقيق السابق تم الاستعانة بمحاكي Cooja، يعمل على نظام تشغيل Contiki-NG، وهو عبارة عن محاكي شبكي مفتوح المصدر يستخدم في أغلب الدراسات البحثية المتعلقة بإنترنت الأشياء، بسبب دعمه للعديد من أنواع الأشياء التي يمكن استخدامها في تطبيقات إنترنت الأشياء الحديثة، كما يحتوي على دعم لأغلب البروتوكولات المستخدمة في طبقات IoT. وبالتالي التحقيق الذي يقدمه البحث يقدم إضافة لنظام تشغيل Contiki-NG يمكن استخدامه في بناء تطبيقات باستخدام المحاكي cooja.



الشكل (8) سيناريو إجراء التجارب

تم تقييم الأداء بالاعتماد على سيناريو واحد كما هو موضح في الشكل (8). يتألف هذا السيناريو من مخدم وزبون وموجه حدودي ((Border Router (BR)). يتعامل المخدم مع رسالة كاملة ذهاباً وإياباً. يقوم الزبون بإرسال 20 رسالة بطلب get إلى مورد موجود على المخدم (رسالة hello) عبر الموجه الحدودي وانتظار الاستجابة منه. تم تقييم أداء السيناريو بخمس حالات:

1- CoAP: يتم الارسال والاستقبال بدون وجود أي حماية. تمت الاستفادة من تحقيق بروتوكول CoAP في نظام تشغيل Contiki-NG.

2- CoAP + DTLS: يقوم الزبون بإرسال الطلبات محمية باستخدام بروتوكول DTLS، مع تجاهل وجود مصافحة handshake في هذه الحالة، وإنما البدء بالحساب بعد الانتهاء من المصافحة. تمت الاستفادة من مكتبة tinyDTLS في نظام تشغيل Contiki-NG.

3- CoAP + OSCORE: يقوم الزبون بإرسال الطلبات محمية باستخدام بروتوكول OSCORE، يستقبل المخدم الطلب المشفر ويفك تشفيره ويقوم بارسال الرد إلى الزبون. في هذه الحالة لايوجد أي بروتوكول لتبادل المفاتيح ما بين الزبون والمخدم.

4- CoAP + DTLS مع مصافحة: يتم إجراء مصافحة مابين الزبون والمخدم قبل أن يتم تبادل الطلبات وحمايتها باستخدام بروتوكول DTLS.

5- OSCORE + EDHOC: يتم تبادل المفاتيح باستخدام بروتوكول EDHOC ليتم بعدها تبادل الطلبات بين الزبون والمخدم. تم تطبيق التحقيق في هذا البحث.

بما أن عملية واحدة مثل حماية رسالة CoAP تأخذ فقط عدة ميلي ثانية، بالإضافة إلى أن النتائج في أنظمة تشغيل الزمن غير الحقيقي يمكن أن تتأثر بالانتهاء من عمليات أو نشاطات موجودة في الخلفية. تم ايجاد حل لهذه المشكلة عن طريق أن المهمة لاتعمل فقط مرة بل عدة مرات وزمن التنفيذ الكلي يمكن أن يقسم على عدد مرات التنفيذ، وهذا يعطي معدل للقيم أكثر دقة. ولذلك في الحالات الخمسة السابقة أعيدت التجارب 5 مرات ومن ثم أخذ حساب المتوسط.

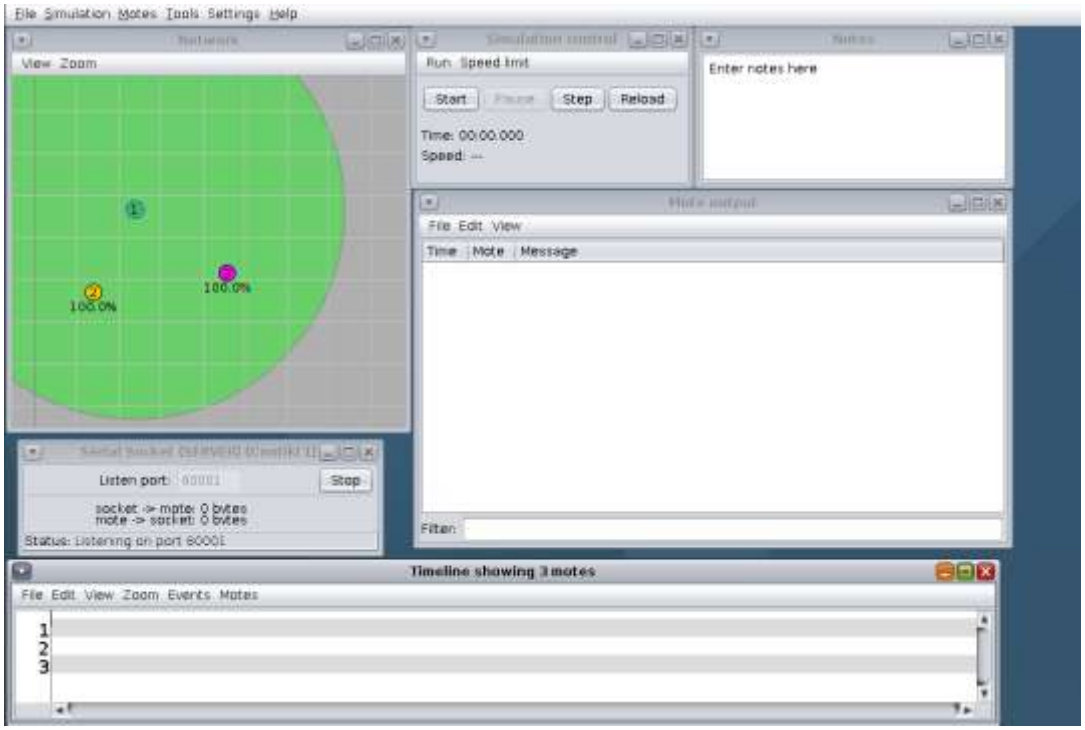
يعرض الشكل (9) واجهة المحاكي Cooja والتي تحتوي الكثير من الأقسام أهمها:

1- قسم الشبكة والذي يعرض العقد التي تتألف منها شبكة إنترنت الأشياء. كل عقدة عبارة عن محاكاة لجهاز حقيقي يستخدم في شبكة إنترنت الأشياء. يوجد الكثير من الأجهزة التي يدعمها Cooja. كل عقدة في هذه الشبكة تدعى mote. تم استخدام جهاز Cooja mote في التقييم.

يوجد ثلاثة أجهزة في الشكل، واحدة من أجل عقدة الزبون والثانية من أجل الموجه الحدودي والثالثة من أجل المخدم.

2- قسم التحكم بالمحاكاة Simulation control: يتم بدأ المحاكاة وايقافها وإعادة تحميلها بالإضافة إلى عرض الوقت الذي استغرقته عملية المحاكاة.

3- الخرج Mote output ويعرض الرسائل التوضيحية المستخدمة في الأكواد والتي تكون خرج جميع الأجهزة خلال عملية المحاكاة.

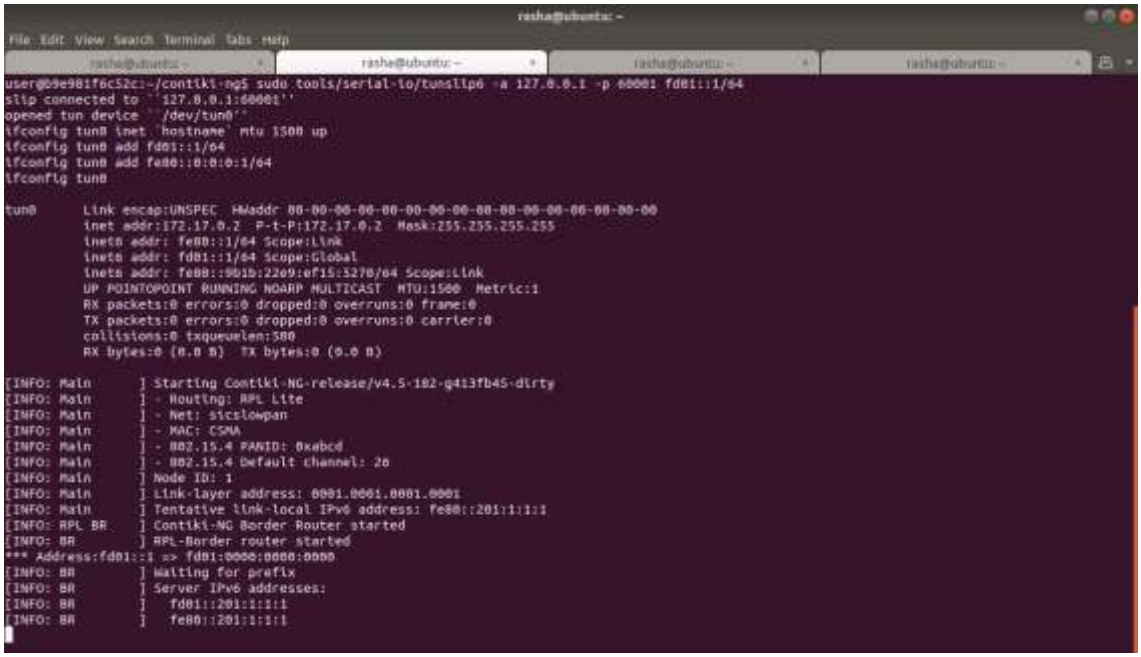


الشكل (9) واجهة Cooja

يساعد الموجه الحدودي في توصيل شبكة بأخرى. في هذا البحث، يتم استخدام BR لتوجيه البيانات بين الشبكة الداخلية الموجودة في محاكي Cooja وشبكة خارجية. حتى الآن قمنا بإنشاء الشبكة الداخلية فقط. نحتاج الآن إلى محاكاة السيناريو الذي تتصل فيه شبكة محاكي Cooja بشبكة خارجية، لهذا الغرض سوف نستخدم أداة Tunslip6 المتوفرة في Contiki-NG. في هذا المثال، ينشئ Tunslip جسرًا بين الشبكة الداخلية والجهاز المحلي الذي يستضيف الشبكة، عن طريق وضع منفذ للموجه الحدودي الموجود في Cooja، كما في المثال 60001 ومن ثم من نفس docker image التي يعمل عليها نظام تشغيل Contiki-NG يجب وضع التعليمة التالية في terminal:

```
sudo tools/serial-io/tunslip6 -a 127.0.0.1 -p 60001 fd01::1/64
```

لينتج لدينا الخرج كما في الشكل (10):

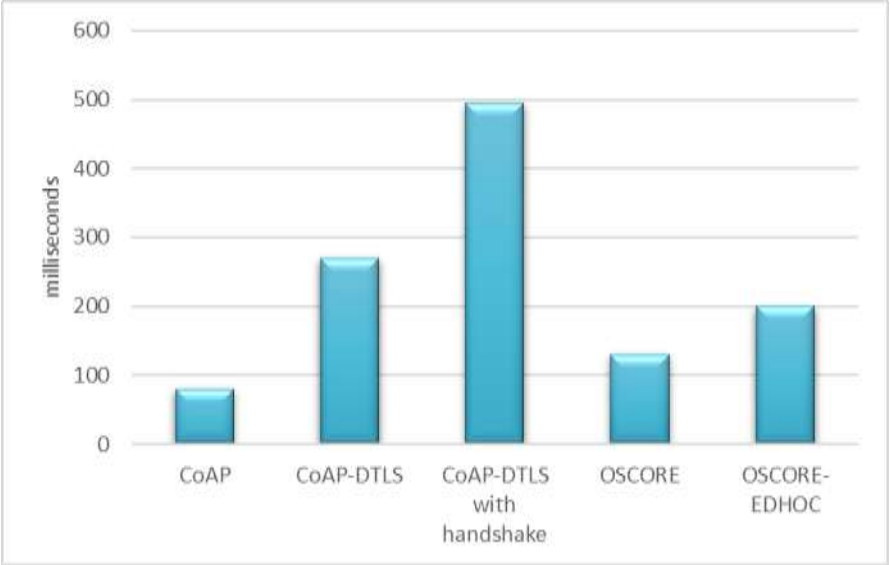


الشكل (10) تفعيل الـ BR

يعتمد نطاق الدراسة بشكل اساسي على اداء الشبكه للبروتوكولات في السيناريوهات الخمسه السابقه. وبشكل أكثر تحديدًا، فإن قياسات الأداء المستخدمة في تقييم السيناريوهات هي التأخير والإنتاجية. تعمل الإنتاجية على تقييم عدد الرسائل بالثانية التي تم تسليمها أو توصيلها بنجاح، بينما يعرف التأخير بتقييم الوقت اللازم لتسليم 20 رسالة بنجاح.

مخطط التأخير Latency: تشبه قيمة التأخير المحسوبة قيمة زمن الرحلة الانكفائية (round-trip time (RTT، في هذه الحالة يقوم الزبون بإرسال 20 رسالة get إلى المخدم و قياس وقت تلقي جميع الاستجابات منه بنجاح بالميلي ثانية ms. نلاحظ من الشكل(11) والذي يظهر أن تطبيق أي نوع من الحماية سوف يزيد من قيمة التأخير الحاصل بشكل كبير مقارنة بنفس السيناريو لاستخدام بروتوكول CoAP من دون حماية، ولكن تختلف قيمة التأخير بحسب البروتوكول المستخدم للحماية، ففي حال استخدام DTLS سيكون التأخير أكبر من حالة استخدام بروتوكول OSCORE. وبالتالي استخدام بروتوكول OSCORE يعد أفضل للاستخدام كحماية من بروتوكول DTLS حتى في حال عدم حساب وقت مصافحة DTLS ووقت بروتوكول EDHOC، وذلك لأن DTLS يقوم بتشفير كل حزمة بروتوكول CoAP في طبقة النقل، بينما يعمل بروتوكول OSCORE في طبقة التطبيقات بتشفير أجزاء محددة فقط من حزمة CoAP.

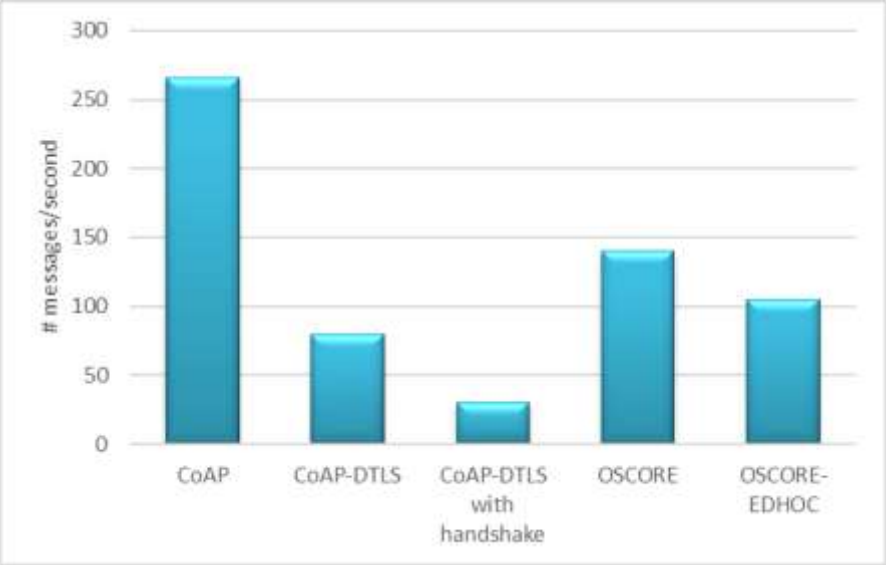
كما نلاحظ أن استخدام EDHOC لا يؤثر بشكل كبير على أداء OSCORE الكلي. في حين وجود مصافحة في بروتوكول DTLS تؤدي إلى زيادة التأخير بشكل ملحوظ، يعود ذلك إلى أن المصافحة تتألف من عدة مراحل، يتم فيها تبادل المفاتيح عن طريق ارسال الرسائل المشفرة في 7 رحلات مابين الزبون والمخدم [14]، وهذا يؤدي بدوره إلى تأخير كبير، في حين بروتوكول EDHOC يقتصر على ثلاثة رسائل بين الزبون والمخدم تستخدم لتبادل المفاتيح. وهذا يوضح سبب الفرق فيما بينهما.



الشكل (11) مخطط التأخير

مخطط الإنتاجية Throughput: يعرض الرسم البياني في الشكل (12) معدل عدد الرسائل المستلمة في عقدة الزبون في الثانية، أي أن الزبون يرسل طلبات إلى المخدم عن طريق BR ويستقبل استجابة ويحسب العدد. تكشف النتائج أيضًا عن العبء لمصافحة DTLS. وذلك لأن مصافحة DTLS الأولية تتطلب رحلات ذهاب وإياب إضافية قبل إنشاء الاتصالات، فبروتوكول DTLS يحتوي طبقتين (طبقة المصافحة وطبقة التسجيل)، طبقة المصافحة تحتوي على أربع بروتوكولات فرعية (المصافحة Handshake ، تغيير مواصفات التشفير change cipher spec ، التنبيه Alert ، بروتوكولات التطبيق Application)، في حين أن طبقة التسجيل تحتوي على بروتوكول التسجيل [14]، والتي بدورها تؤثر سلبًا على البروتوكولات المستخدمة جنبًا إلى جنب معها. في المقابل، تبقى بصمة EDHOC صغيرة لجميع القياسات.

من خلال ماسبق نستنتج أن أداء OSCORE أفضل من CoAP عبر DTLS. بالإضافة إلى أن استخدام بروتوكول تبادل المفاتيح خفيف الوزن EDHOC أدى إلى زيادة في الأداء ولكن هذا طبيعي مقارنة بالمهمة المطلوب القيام بها. في الواقع، هناك عوامل أخرى يمكن أن تؤثر على عبء رسالة OSCORE مثل عدد الخيارات المشفرة في CoAP. ومع ذلك، تم النظر في هذا العامل في تقييمنا. تضمنت جميع رسائل CoAP نفس خيارات CoAP في جميع القياسات لتحقيق أفضل النتائج الممكنة.



الشكل (12) مخطط االإنتاجية

الاستنتاجات والتوصيات:

قمنا في هذا البحث بتطبيق حماية على بروتوكول التطبيقات المقيدة CoAP باستخدام بروتوكول OSCORE، بالإضافة إلى تطبيق بروتوكول EDHOC لتبادل المفاتيح في نظام تشغيل Contiki-NG. وتقييم أداء التحقيق باستخدام محاكي Cooja ومقارنته باستخدام بروتوكول CoAP في سيناريو غير آمن، مع استخدام بروتوكول CoAP في سيناريو آمن بحمايته عن طريق بروتوكول DTLS. تم تقييم الأداء نسبة للتأخير والإنتاجية. أظهرت النتائج أن تطبيق الحماية يزيد التأخير ويقلل الإنتاجية وهذا شيء متوقع نسبة للفائدة المرجوة، ولكن استخدام بروتوكول OSCORE أفضل من استخدام بروتوكول DTLS. كما قام البحث بمقارنة العبء الذي يفرضه مصافحة DTLS وتبادل مفاتيح بروتوكول OSCORE باستخدام بروتوكول EDHOC، تبين أن استخدام بروتوكول EDHOC لتبادل المفاتيح يعطي عبء أقل من مصافحة DTLS وهذا يعطي أولوية لاستخدامه في تطبيقات إنترنت الأشياء، وخاصة أن التطبيقات تسعى لاستخدام البروتوكولات التي تعطي دائماً عبء قليل.

إن تقييم مقاييس الأداء الأخرى ذات الصلة (مثل استهلاك الطاقة أو استخدام الموارد كذاكرة ومعالجة)، أي دراسة العبء الذي يفرضه استخدام الحماية على العقد، يمكن أن يوفر رؤى جديدة حول فعالية بروتوكولات أمان إنترنت الأشياء المختلفة. ومع ذلك، فإن تقييم مقاييس الأداء هذه متروك للدراسات البحثية المستقبلية. الجدير بالذكر، أن تطوير بروتوكول DTLS 1.3 أيضاً يمثل تطلعاً مستقبلياً [19] ، هذا البروتوكول قيد التصميم من قبل IETF ولا يوجد تحقيقات حالية له، لكن من المفترض أن يؤدي هذا البروتوكول إلى تقليل زمن الانتقال وزيادة الإنتاجية في نقل البيانات.

References:

[1] Gunnarsson, M. *Security Solutions for Constrained Devices in Cyber-Physical Systems*.Master Thesis, Lund University, 2020.

[2] Selander, G.; Mattsson, J.; Palombini, F.; & Seitz, L. *Object security for constrained restful environments (oscore)*, RFC (7252), 2019.

[3] Gunnarsson, M.; Brorsson, J.; Palombini, F.; Seitz, L.; & Tiloca, M. *Evaluating the performance of the OSCORE security protocol in constrained IoT environments*. 2021 Internet of Things, volume 13, 100333. ISSN 2542-6605.

[4] Selander, G., Mattsson, J., & Palombini, F. *Ephemeral Diffie-Hellman Over COSE (EDHOC) draft-ietf-lake-edhoc-12*, 10Jun. 2022. <https://datatracker.ietf.org/doc/html/draft-ietf-lake-edhoc-12>.
[5] Bormann, C., and P. Hoffman. "*RFC 8949 Concise Binary Object Representation (CBOR)*." (2020).
[6] Schaad, Jim. *Cbor object signing and encryption (cose)*, (RFC 8152) 2016. < https://www.hjp.at/doc/rfc/rfc8152.html>
[7] Dandah R., Ahmad A., Ghadeer R. , *A Study of Using DTLS and CoAP protocols in the secure transfer of images in a cloud-based IoT system*. Tuj-eng [Internet]. 2021Sep.14 [cited 2022Feb.20];43(4). (In Arabic)
[8] Kantharajan, K. ; and Shirafkan, S. ; *Efficient Security Protocol for RESTful IoT devices*. Master Thesis, Lund University SE-221 00 Lund, Sweden. 2020.
[9] Disch, M. *Lightweight Application Layer Protection for Embedded Devices with a Safe Programming Language*. Diss. Ph. D. dissertation, Software Engineering Group Department of Informatics, Switzerland, 2020.
[10] von Raumer, M. *Continuous integration of embedded security software*. (2020). Master thesis, University of Fribourg (Switzerland).
[11] Hristozov, S.; Huber, M.; Xu, L.; Fietz, J.; Liess, M.; & Sigl, G. *The Cost of OSCORE and EDHOC for Constrained Devices*. In Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy, (2021, April) , (pp. 245-250).
[12] *CONTIKI-NG*, 10Jun. 2022. <https://www.contiki-ng.org/>.
[13] SHELBY, Z.; HARTKE, K.; BORMANN, C. *The constrained application protocol (CoAP)*. Internet Engineering Task Force (IETF), ISSN: 2070-1721, June. 2014, < https://tools.ietf.org/html/rfc7252>
[14] Eric Rescorla and Nagendra Modadugu, *Datagram transport layer security version 1.2*". 2012. (RFC 6347) < https://www.hjp.at/doc/rfc/rfc6347.html >
[15] Palombini , F. , Tiloca, M. , Höglund, R. , Hristozov, S. , Selander, G. , *Profiling EDHOC for CoAP and OSCORE* , 2021. <https://datatracker.ietf.org/doc/draft-ietf-core-oscore-edhoc/>.
[16] *Contiki-NG Github Repository* , 10Jun. 2022. <https://github.com/contiki-ng/contiki-ng>.
[17] *install Contiki-NG OS using Docker* , 10Jun. 2022. < https://github.com/contiki-ng/contiki-ng/wiki/Docker>.
[18] *OSCORE implementation in Contiki-NG* , 10Jun. 2022. <https://github.com/Gunzter/contiki-ng/tree/master> .
[19] Rescorla, E.; Tschofenig, H.; , Modadugu, N. *The Datagram Transport Layer Security (DTLS) Protocol Version 1.3 draft-ietf-tls-dtls13-43* , 2021. <https://datatracker.ietf.org/doc/draft-ietf-tls-dtls13/>.