Proof-carrying messages for the Internet of Things

Jurriaan Declerc , **Rene Fekkes**
Departement Computerwetenschappen, KU Leuven
25 mei 2020

Abstract

Het Internet of Things (IoT) is een snel groeiend netwerk van apparaten die onze omgeving meten en besturen. Hoewel bedrijven, individuen en gemeenschappen steeds afhankelijker worden van deze apparaten, laat hun beveiliging vaak nog te wensen over.

Remote attestation is een methode om de interne staat van een IoT apparaat te controleren. Door deze methode periodiek uit te voeren, kan een bepaalde zekerheid verkregen worden over het al dan niet malwarevrij zijn van het apparaat. De frequentie waarmee deze methode wordt uitgevoerd heeft een grote invloed op de verkregen betrouwbaarheidsgraad.

In deze paper onderzoeken we een variant van remote attestation waarbij ieder bericht dat verstuurt wordt door een IoT apparaat, een token bevat dat aangeeft of het apparaat al dan niet malwarevrij is. Meer specifiek onderzoeken we de mogelijkheid om deze *Proof-Carrying* messages te realiseren op klasse-0 IoT apparaten. Hiertoe implementeren we een sensor in verschillende applicaties met toenemende gradaties van beveiliging en evalueren we de overhead in termen van geheugenverbruik en uitvoeringstijd. We concluderen dat proof-carrying messages voor klasse-0 apparaten niet voor elke situatie geschikt is omwille van de overhead in termen van rekentijd en energieverbruik.

1 Inleiding

Het Internet of Things (IoT) is een zeer snel groeiend netwerk dat bestaat uit miljoenen apparaten. Deze hebben gemeenschappelijk dat ze verbonden zijn met een netwerk waarop ze autonoom, dus zonder menselijke interactie, data kunnen overbrengen en verwerken. Met een breed scala aan toepassingen waaronder rookmelders, Track & Trace en smart-tv's is beveiliging van groot belang. De grootste deelgroep van deze apparaten heeft bescheiden hardware in vergelijking met de persoonlijke computer. Bovendien wordt de kostprijs van deze apparaten zoveel mogelijk gedrukt, waardoor beveiliging-specifieke hardware zelden aanwezig is.

Een bestaande methode om de integriteit van een IoT toestel te verifiëren, is remote attestation. Francillon et al. definieert remote attestation als een security service die een betrouwbare partij (verifier) toelaat om de interne staat van een onbetrouwbaar ingebed apparaat (prover) vanop afstand na te gaan [6].

2 Remote Attestation

Een welbekende methode om de correctheid van een IoT apparaat te verifiëren, is remote attestation. Remote attestation is een challenge-response protocol waarmee een betrouwbare partij, de *verifier*, de interne staat van een ander apparaat, de *prover*, vanop afstand kan nagaan. Het is dus een techniek waarmee er gecontroleerd kan worden of een apparaat malware bevat, zonder daarvoor in de fysieke nabijheid van dat apparaat te moeten zijn.

Bij deze methode zijn twee partijen betrokken:

- **Prover**: Het apparaat waarvan men wilt weten of het geen ongewenste software uitvoert. We noteren de staat van de prover met S_{Prov}.
- **Verifier**: Het apparaat dat de staat van de prover wil nagaan. De verifier verwacht dat de prover in een bepaalde staat S_{Ver} is. Als de verifier detecteert dat $S_{Prov} \neq S_{Ver}$, weet deze dat de prover geïnfecteerd is.

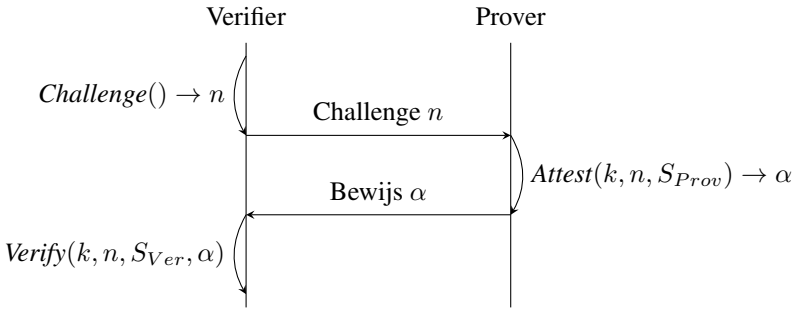
Het remote attestation protocol zoals gedefinieerd door Francillon et al. in [6] bestaat uit drie algoritmes:

1. $Setup(1^{\mathcal{K}})$: Een algoritme dat, gegeven een parameter $1^{\mathcal{K}}$, een gedeelde sleutel k produceert.
2. *Attest*(k, S): Een algoritme dat, gegeven een sleutel k en staat S, een attestatie token α produceert.
3. *Verify*(k, S, α): Een algoritme dat, gegeven een sleutel k en staat S, de booleaanse waarde *true* produceert als en slechts als $Attest(k, S) = \alpha$.

In wat volgt werken we met een uitbreiding van *Attest* en *Verify*, waarbij er ook een nonce n wordt meegegeven als input.

In de rest van deze sectie doorlopen we de uitvoering van het remote attestation protocol, dat grafisch wordt voorgesteld

in figuur 1. We gaan ervan uit dat *Setup* al is uitgevoerd en dat er dus een gedeelde sleutel k bestaat tussen de prover en verifier.



Figuur 1: Het remote attestation protocol

De verifier initieert het remote attestation protocol door het genereren van een challenge n. Deze functioneert als een nonce en zal ervoor zorgen dat het resultaat van $Attest(k, n, S)$ zelfs bij een gekende k en S, onvoorspelbaar is. Door iedere keer een willekeurige nonce te genereren, worden precomputation en replay attacks vermeden. De challenge n wordt doorgestuurd naar de prover.

De prover zal vervolgens het *Attest* algoritme uitvoeren met als inputs de gedeelde sleutel k, de ontvangen nonce n en zijn huidige staat S_{Prov} om een bewijs α te genereren. Steiner et al. beargumenteert in [10] dat de implementatie van *Attest* aan enkele eigenschappen moet voldoen:

- Authenciteit: De verifier moet kunnen kunnen afleiden dat het bewijs α door de prover is opgesteld.
- Atomiciteit: De attestatieroutine moet atomisch worden uitgevoerd zodat eventuele malware niet in staat is om ongedetecteerd te blijven door de routine te onderbreken.
- Onvervalsbaarheid: Het moet voor een aanvaller dermate lang duren om het bewijs α te bekomen, dat dit detecteerbaar is voor de verifier.
- Determinisme: Opdat de verifier een gegeven bewijs kan nakijken, is het noodzakelijk dat $\alpha = Attest(k, n, S_{Ver})$ als en slechts als $S_{Ver} = S_{Prov}$.

Een mogelijke implementatie van *Attest* zou α kunnen berekenen als S XOR k XOR n. De representatie van S kan echter zodanig veel geheugen in beslag nemen, dat het onpraktisch wordt om α te verzenden over een netwerk met een lage bandbreedte. Daarom zal het bewijs α een vaste grootte hebben die beduidend kleiner is dan S.

Het is essentieel dat *Attest* door de Trusted Platform Module (TPM) van de prover wordt berekend. Een TPM is een hardware- of software component die een anker van vertrouwen vormt. We gaan er van uit dat een TPM volledig veilig is. Data die wordt opgeslagen in een TPM, zoals geheime sleutels, mag op geen enkele manier beschikbaar zijn voor andere onderdelen van het systeem. Mocht *Attest* niet geïmplementeerd zijn in een TPM, zou het mogelijk zijn dat een geïnfecteerd apparaat alsnog een correct bewijs levert dat echter niet de interne staat van de prover weerspiegelt. Een aanvaller zou namelijk de Attest methode kunnen onderbreken en telkens delen van de interne staat kunnen herstellen zodat zijn malware ongedetecteerd blijft. Doordat *Attest* geplaatst wordt in een TPM wordt atomiciteit gegarandeerd en is het voor een aanvaller niet mogelijk om de beschreven aanval uit te voeren.

Remote attestation steunt zeer sterk op de veiligheid die de Trusted Platform Module aanbiedt, waardoor het protocol hiervan sterk afhankelijk is. Als mocht blijken dat de TPM feilbaar is, zal remote attestation niet meer betrouwbaar zijn.

Nadat de prover een bewijs α heeft gegenereerd door middel van de *Attest* methode, wordt die als response naar de verifier verstuurd.

Ten slotte zal de verifier $Verify$ uitvoeren met als inputs de gedeelde sleutel k, de eerder gegenereerde nonce n, de verwachte staat van de prover S_{Ver} en het ontvangen bewijs α. Zoals eerder vermeld zal de verifier in feite $Attest$ uitvoeren, maar dan met S_{Ver} in plaats van S_{Prov}. Als de output daarvan overeenkomt met de ontvangen token α, weet de verifier met een bepaalde zekerheid dat de prover zich in een correcte staat bevindt.

Om te voorkomen dat een geïnfecteerd apparaat het genereren van een bewijs afwentelt op andere (krachtigere) computers, analyseert de verifier ook hoelang een prover er over doet om een bewijs op te stellen [10]. Dit valt echter buiten het bestek van deze paper.

Ten slotte staan we stil bij de mogelijks verwarrende gebruik van het woord *bewijs*. In de context van remote attestation slaat de term op data die onweerlegbaar aantoont dat een apparaat zich al dan niet in de verwachte staat bevindt. Echter, de manier waarop de staat van een apparaat wordt voorgesteld kan in meer of mindere mate allesomvattend zijn. Remote attestation laat een verifier toe om een bepaalde graad van vertrouwen toe te kennen aan een prover, maar geen volledige zekerheid. Daarom spreekt men in deze context vaak over integriteitsmetingen [4]. Vastleggen welke eigenschappen van een apparaat worden opgenomen in de staat is een afweging tussen volledigheid en de haalbaarheid voor de verifier om deze staat te kunnen verifiëren. Zo wordt er vaak gekozen om de inhoud van het dynamisch geheugen van een prover niet mee op te nemen in de staat, al is hier wel onderzoek naar gedaan[8] [11].

3 Probleemstelling

In deze sectie bespreken we een zwakte van remote attestation, en formuleren we een mogelijke oplossing.

Een probleem met het remote attestation protocol is dat de het genereren van een bewijs op basis van een zinvolle voorstelling van de staat vaak enige tijd duurt. Een eerste manier om hier mee om te gaan is om de prover bij de eerste uitvoering van *Attest*, het berekende bewijs op te laten slaan. Wanneer de prover nadien opnieuw wordt gevraagd om een integriteitsbewijs te leveren, stuurt die simpelweg het

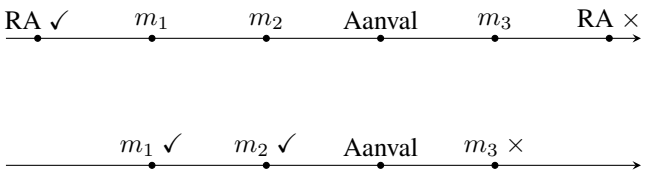
opgeslagen bewijs terug.

Deze aanpak heeft twee grote zwaktes:

1. Deze strategie laat het gebruik van nonces niet toe. Het doel van een nonce is immers om iedere keer een ander bewijs te krijgen, ook als alle andere omstandigheden gelijk blijven.
2. Met deze strategie kan enkel bewezen worden dat het apparaat zich initieel (niet) in een correcte staat bevond. Als er nadien een aanval plaatsvindt, zal dat niet gedetecteerd worden.

Het is om deze redenen noodzakelijk dat de prover bij iedere uitvoering van het remote attestation protocol, een nieuw bewijs genereert. Een tweede mogelijke strategie is dat de verifier periodiek een challenge stuurt en de prover telkens een nieuw bewijs genereert. Hoe vaak dit precies gebeurt is een afweging tussen de graad van beveiliging en de last op de prover. Voor apparaten die werken op een batterij, kan het extra energieverbruik ten gevolge van het *Attest* algoritme een niet-verwaarloosbare impact hebben op de levensduur van het apparaat.

Hoewel minder vaak attesteren een positief effect heeft voor het energieverbruik, wordt in figuur 2 geïllustreerd hoe dit ten nadele van de veiligheid gaat.



Figuur 2: Tijdslijn van remote attestation (boven) en proof-carrying messages (onder)

Op deze figuur zien we dat de verifier initieel vaststelt dat de prover zich in een correcte staat bevindt met behulp van remote attestation (RA). Vervolgens ontvangt de verifier berichte m_i. Tussen bericht m_2 en m_3 wordt de prover succesvol aangevallen. Enige tijd nadat m_3 wordt ontvangen, voert de verifier opnieuw remote attestation uit en detecteert dat de prover is aangevallen. De verifier kan niet weten dat de prover nog niet was aangevallen op het moment dat m_1 en m_2 werden verstuurd. Daarom zou de verifier niet mogen handelen op basis van een van de drie ontvangen berichten. Als het apparaat deel uitmaakt van feedback loop kan dit problematisch zijn: het systeem moet tijdig feedback krijgen (en kan dus niet wachten tot RA drie berichten tegelijk goedkeurt), maar het is ook niet accepteerbaar dat het systeem handelt op wat later een frauduleus bericht blijkt te zijn.

Een voor de hand liggende oplossing voor dit probleem is om de verifier RA uit te laten voeren voor ieder bericht dat ontvangen wordt. Deze situatie wordt onderaan in figuur 2 getoond. In dit geval is er een zekere redundantie in de communicatie. Als de prover weet dat de verifier meteen om een bewijs zal vragen wanneer het een bericht ontvangt, kan hij proactief een bewijs genereren en dit tezamen met het bericht versturen. Zo een bericht noemen we een proof-carrying message.

Een grote uitdaging bij de implementatie van proof-carrying messages op IoT apparaten is hun beperkte hardware; zowel in termen van geheugen als rekenkracht. De Internet Engineering Task Force (IETF) deelt IoT apparaten op in drie klassen op basis van hun geheugen zoals weergegeven in tabel 1 [7].

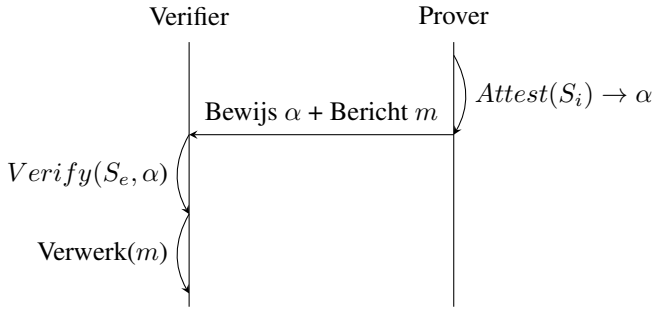
In deze paper onderzoeken we of het mogelijk is om proof-carrying messages te implementeren op klasse-0 apparaten en zo ja, voor welke toepassingen dit relevant kan zijn.

	Data geheugen	Instructie geheugen
Klasse-0	<<10 kB	<<100 kB
Klasse-1	±10 kB	±100 kB
Klasse-2	±50 kB	±250 kB

Tabel 1: De classificatie van IoT apparaten op basis van hun beschikbaar geheugen zoals beschreven in RFC 7228 [7]

4 Proof-Carrying messages

Proof-carrying messages is een variant van het remote attestation protocol waarbij er niet gevraagd wordt om een bewijs, maar deze wordt telkens meegeleverd met ieder bericht. Het verbeterd de betrouwbaarheid van berichten doordat de ontvanger een inzicht krijgt in welke staat de prover zich bevond tijdens de opstelling van de boodschap.



Figuur 3: Het proof-carrying messages protocol

Dit protocol bestaat uit de volgende drie algoritmes die overeenkomen met diegenen van Remote Attestation.

1. $Setup(1^{\mathcal{K}}) \rightarrow k$
2. $Attest(k, S) \rightarrow \alpha$
3. $Verify(k, S, \alpha) \rightarrow boolean$

Het initiëren van dit protocol door de verifier met de besproken Challenge() is niet meer van toepassing. In plaats hiervan zal de prover autonoom een bewijs opstellen wanneer deze een bericht verstuurt naar de verifier. Ook in deze variant zal de attestatie functie zich bevinden in de TPM. De eigenschappen determinisme, authenticiteit en atomiciteit worden alsnog gegarandeerd. Om aan onvervalsbaarheid te voldoen moet er echter een kleine verandering gebeuren. Hiervoor werd deze deels verzekert met behulp van de nonce die geven werd in de challenge n. Doordat de prover bij

proof-carrying messages geen challenge ontvangt van de verifier zal deze er zelf een moeten opstellen en meesturen met het bewijs. Deze nonce generatie wordt uitgevoerd door de TPM. Daarnaast is er geen timing mogelijk doordat de verifier niet weet wanneer de prover begonnen is met het bewijs op te stellen. Bij remote attestation werd als startpunt de Challenge methode gebruikt. Het opstellen van een vals bewijs is alsnog niet mogelijk doordat een bewijs altijd legitiem wordt opgesteld door de TPM. Een mogelijkheid om dan toch valse bewijzen te gebruiken is imitatie. Dit is enkel mogelijk indien een derde partij de geheime sleutel bemachtigd heeft. Dit geval wordt niet verondersteld omdat er vanuit wordt gegaan dat de TPM betrouwbaar is.

Proof-carrying messages heeft enkele voordelen ten opzichten van remote attestation. De berichten van de prover zijn zoals besproken betrouwbaarder. Dit komt omdat de verifier weet in welke staat de prover zich bevond tijdens het verzenden van het bericht. Hierdoor verwerkt de verifier geen valse berichten meer en kan er met een hogere zekerheid gehandeld worden. Daarnaast vereist deze in tegenstelling tot de klassieke attestatie routine geen downstream berichten. Dit kan een handige eigenschap zijn voor situaties waarbij dit vermeden moet worden. Er zijn echter ook enkele nadelen. Er is een overhead die niet meer flexibel in te stellen valt als bij remote attestation. Daarbij kon de verifier zelf de routine frequentie verlagen. De routine frequentie bij proof-carrying messages hangt volledig af van de frequentie van verzonden berichten van de prover. Daarbovenop is er een nieuwe aanval mogelijk. Het versturen van berichten van een toestel kan volledig uitgeschakeld worden. Indien er enkel bewijzen worden verstuurd met berichten zal de verifier er nooit achter komen dat de staat van de prover verandert is. Ten slotte zal het opstellen van een bewijs telkens een vertraging geven bij het verzenden van berichten. Indien de verifier zekerheid wil zal deze dan ook eerst de verify methode moeten uitvoeren vooraleer hij handelt. Dit kan voor tijdskritische situaties voor problemen zorgen.

5 Werkwijze

Om aan te tonen dat het proof-carrying messages protocol te verwezenlijken is op klasse-0 IoT apparaten, is er een proof-of-concept applicatie met toenemende gradaties van beveiliging ontwikkeld [1].

- **Referentie**: Een applicatie die op regelmatige tijdstippen de waarde van een sensor leest en verstuurt via een seriële verbinding.
- **Versleuteling**: Een uitbreiding van de referentie applicatie waarbij het bericht versleutelt wordt.
- **Proof-Carrying**: Een variant van de versleuteling applicatie waarvan het bericht ook een integriteitsbewijs bevat.
- **TPM**: Een versie van de proof-carrying applicatie die gebruikt maakt van een TPM.

In het vervolg van deze sectie worden de belangrijkste ontwerpkeuzes belicht.

[1] De code is beschikbaar via https://github.com/JurriaanD/Proof-Carrying-Messages

5.1 Klasse-0 IoT apparaat

De applicaties werden ontwikkelt voor de Arduino Uno rev 3. Uit tabel 1 en tabel 2 met de specificaties van de Arduino, kan afgeleid worden dat dit daadwerkelijk een klasse-0 IoT apparaat is. Het is belangrijk om in te zien dat er geen ondergrens is voor klasse-0 apparaten. Daarom zal deze paper geen uitspraken doen over de deelgroep van klasse-0 apparaten met mindere capaciteiten dan de gebruikte Arduino.

	Specificaties
Microcontroller	ATmega328P (8 bit)
Flash Memory	32 kB
SRAM	2 kB
EEPROM	1 kB
Kloksnelheid	16 MHz

Tabel 2: De specificaties van de Arduino Uno rev 3 [2]

5.2 Cryptografie

(A)symmetrische encryptie

Om proof-carrying messages te implementeren, moet er een versleutelingstechniek worden gekozen. Het voordeel van berichten versleutelen is tweevoudig: de authenticiteit van de verzender kan geverifieerd worden en andere partijen kunnen de verstuurde data niet onderscheppen.

Een eerste keuze die gemaakt moet worden, is of er een symmetrische dan wel een assymetrische versleutelingstechniek gebruikt wordt.

Asymmetrische encryptie heeft als voordeel dat het toelaat om versleutelde berichten uit te wisselen zonder dat er op voorhand een gedeeld geheim moet gekozen worden. Daartegenover staan echter twee grote nadelen: vergeleken met symmetrische sleutels zijn de asymmetrische in het algemeen langer en de benodigde cryptografische operaties vergen meer rekenkracht. Aangezien we op zoek zijn naar een cryptografische techniek die goed op resource-constraint apparaten werkt, wegen de nadelen van asymmetrische versleuteling sterker door. Daarnaast is het vermelde voordeel niet zozeer relevant voor het huidige gebruik van proof-carrying messages. Er kan immers vanuit gegaan worden worden dat het IoT toestel meestal met slechts één entiteit, een verifier, zal communiceren.

Sleuteluitwisseling

De keuze voor symmetrische encryptie betekent dat er ook een manier moet worden voorzien voor de prover en verifier om een symmetrische sleutel te kiezen. Het uitwisselen van sleutels kan ofwel vooraf gebeuren in een veilige omgeving, ofwel nadat het apparaat al actief is door middel van een sleuteluitwisselingsprotocol. Een welbekende techniek is het Diffie-Hellman protocol, dat twee partijen toelaat om een gedeeld geheim uit te wisselen over een onbeveiligd kanaal. Een probleem met het protocol zoals het oorspronkelijk in 1976 werd voorgesteld, is dat het

steunt op het factorisatieprobleem en dus gebruikt maakt van grote priemgetallen. Op het moment van schrijven is het aangeraden om priemgetallen van 2048-3072 bits te gebruiken. Een variant van DH, Elliptische Curve Diffie-Hellman, werkt op basis van het discrete logaritme probleem. De aanbeveling voor de lengte van een sleutel van deze variant is slechts 256 bits. Een probleem waar beide versies onder lijden, is dat de implementatie van de algoritmen tamelijk complex is. De kleinste implementatie die we konden vinden voor het 8-bit AVR platform voegt ongeveer 7kB toe aan een programma. In de context van resource-constrained apparaten is dit een belangrijke factor. Daarnaast doet deze implementatie er iets meer dan acht seconden over om een geheim uit te wisselen over een snelle seriële verbinding.

Omwille van deze redenen is er geen sleuteluitwisselingsalgoritme toegevoegd aan de applicaties. Er wordt vanuit gegaan dat symmetrische sleutels vooraf in een veilige omgeving zijn uitgewisseld.

Versleutelingsalgoritme
We kozen voor AES-128 als symmetrisch versleutelingsalgoritme. AES is een veelgebruikt algoritme waardoor er ook veel en performante implementaties beschikbaar zijn.

5.3 Integriteitsbewijs

De ATmega328P maakt, zoals de meeste microcontrollers, gebruik van de Harvard-architectuur. In tegenstelling tot de von Neumann architectuur, is het geheugen voor data (SRAM) en instructies (flash) fysiek gescheiden. Aangezien de inhoud van het instructiegeheugen volledig equivalent is met het programma dat wordt uitgevoerd, is besloten om dit te gebruiken als de interne staat van het IoT apparaat. Een verifier kan eenvoudig een kopie bijhouden van de originele staat als S_Ver, omdat deze statisch is, in tegenstelling tot SRAM en registers.

5.4 Trusted Platform Module

Zoals eerder vermeld, kan een Trusted Platform Module geïmplementeerd worden door middel van een extra microcontroller. Ook een software of een hybride aanpak is mogelijk. Omdat klasse-0 apparaten doorgaans voor een specifieke taak worden ontworpen en de productieprijs zo laag mogelijk wordt gehouden, is er zelden hardware ondersteuning voor een TPM. Om dezelfde reden ontbreekt er een memory protection unit (MPU), waardoor het ontwerpen van een softwarematige aanpak geen eenvoudige zaak is. De afgelopen jaren is er echter veel onderzoek gevoerd naar softwarematige TPMs, waardoor er reeds enkele werkende versies bestaan. We hebben gekozen om gebruik te maken van de Security MicroVisor (SμV) [5].

Security MicroVisor
De SμV zorgt voor geheugen isolatie door middel van software virtualisatie en assembly-level code verificatie. Het geheugen waarin de Security MicroVisor zich bevindt, wordt door de SμV beschermt door de genoemde isolatietechnieken. Op die manier wordt er een stuk virtueel ROM gecreëerd, dat enkel kan worden overschreven bij fysieke toegang.

De hardwarevereisten van de aangepaste SμV zijn minimaal, namelijk een 4 kB flash geheugen en een single threaded CPU. Deze laatste eigenschap is nodig voor de atomisch uitvoering van kritische secties. Hiervoor moet het ook mogelijk zijn om interrupts uit te schakelen. De controller op de Arduino Uno, namelijk de ATmega328P, voldoet aan al deze vereisten.

De SμV voorziet al een attestatie routine die HMAC-SHA1 gebruikt om een hash te genereren van het instructiegeheugen. Hoewel reeds in 2005 werd aangetoond dat het computationeel haalbaar is om een SHA1 collision te veroorzaken [9], wordt HMAC-SHA1 nog steeds gezien als veilig [3].

6 Evaluatie

6.1 Waarnemingen

Twee aspecten worden geëvalueerd: - De tijd die nodig is om een bericht op te stellen. - De grootte van de applicatie. De uitvoeringstijd is relevant omdat dit een maat is voor het aantal benodigde clock cycles. De clock cycles zijn op hun beurt belangrijk omdat klasse-0 apparaten beperkte rekenkracht hebben. Daarnaast wordt als energiebron vaak een batterij gebruikt waardoor zuinig moet worden omgesprongen met energie. Er moet ook rekening worden gehouden met de grootte van de totale applicatie. Aangezien het instructiegeheugen van deze IoT apparaten beperkt is.

	Uitvoeringstijd	Programmagrootte
Referentie	< 1 ms	2 kB
Encryptie	2 ms	6 kB
Proof-Carrying	1143 ms	10 kB
SμV	1143 ms	15 kB

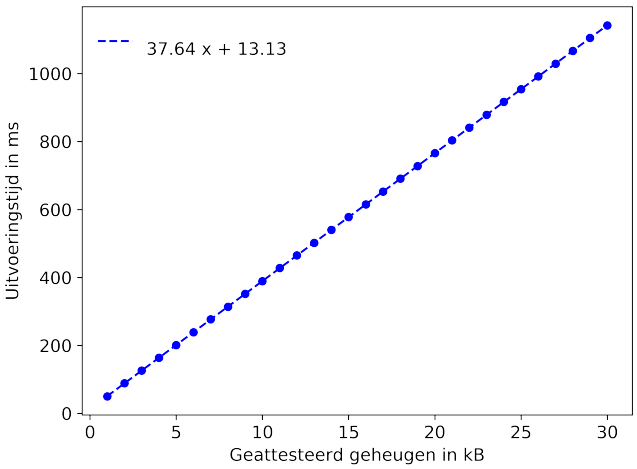
Tabel 3: Resultaten voor de vier ontwikkelde applicaties

In de tabel is te zien dat de referentie applicatie iets minder dan 2 kB groot is. Het grootste deel van de code in deze applicatie bestaat uit methodes om te communiceren door middel van de Universal Asynchronous Transmitter/Receiver (UART).

Bij het toevoegen van encryptie is een kleine toename te zien in zowel de uitvoeringstijd als de grootte van het programma. Een overhead van minder dan 2 ms en et verbruik van 2 kB geheugen zal voor veel toepassingen acceptabel zijn.

Als dit wordt uitgebreid met proof-carrying messages, neemt de uitvoeringstijd toe tot ongeveer een seconde. Hoewel deze waarnemingen gebaseerd zijn op metingen van een specifieke implementatie. Laat de grote orde vermoeden dat proof-carrying messages niet zonder meer toegepast kan worden.

Er is vastgesteld dat bij de toevoeging van de SμV de extra uivoeringstijd minimaal is. De grootte van het programma is echter wel met de helft toegenomen.



Figuur 4: Uitvoeringstijd in functie van de hoeveelheid geattesteerde geheugen

7 Toepassingen

De overhead van proof-carrying messages zorgt ervoor dat de methode niet geschikt is voor elke situatie. Er zijn echter Internet of Things toepassingen waarbij deze extra beveiliging zeer essentieel blijkt en waarbij de performantie minder relevant is.

LoRa [1] is een low-power netwerk dat ontworpen is voor de specifieke noden van IoT apparaten. Het netwerk laat apparaten toe om berichten over lange afstand (10+ km) te versturen met weinig energie. Het netwerk heeft een beperkte bandbreedte tussen de 250 bit/s en 11kbit/s. Daarnaast wordt er aangeraden om niet vaker dan om de vijf minuten een bericht te versturen. In de context van deze aanbeveling lijkt de tijd voor het genereren van een bewijs toch wel mee te vallen.

Een andere interessante eigenschap van dit netwerk is dat het versturen van berichten naar een verbonden apparaat (downlink) zoveel mogelijk vermeden moet worden. Doordat proof-carrying messages een deel downlink communicatie van remote attestation elimineert, is het proof-carrying messages concept goed toe te passen.

Een probleem dat hier wel bij opduikt is dat we spreken over een low-power netwerk. De meeste apparaten die verbonden zijn met het LoRa werken dan ook op een batterij. Voor die apparaten kan de extra energie die verbruikt wordt tijdens het berekenen van een bewijs, een (te) grote kost zijn.

Een ander netwerk voor IoT apparaten is Sigfox. Dit netwerk is beschikbaar in meer dan 70 landen, verspreid over alle continenten. Gelijkaardig aan het LoRa netwerk, kan een verbonden apparaat maximaal zes berichten per uur mag versturen. In die context is de overhead van proof-carrying messages mogelijks aanvaardbaar. Met een maximum van 144 berichten per dag, kijken we bij onze applicatie tegen een tweetal minuten per dag extra tijd dat er berekeningen worden uitgevoerd. Helaas legt het netwerk ook een maximale berichtgrootte van 12 bytes op. In de ontwikkelde proof-of-concept applicaties wordt gebruikt gemaakt van een 20 byte nonce en 20 byte bewijs, wat dus al te groot is. Er zal dus een techniek gevonden moeten worden die kleinere maar nog steeds betrouwbare bewijzen genereert vooraleer de toepassing om dit netwerk mogelijk is.

8 Verder Werk

In deze sectie bespreken we enkele aanpassingen aan en uitbreidingen op de ontwikkelde proof-of-concept applicaties.

8.1 Performantere cryptografie

De Security Microvisor bevat reeds een implementatie van het *Attest* algoritme en produceert een HMAC-SHA1 tag. Een HMAC met een onderliggende hash functie die beter geschikt is voor resource-constrained apparaten zou de performantie van proof-carrying message kunnen verbeteren. Een performanter versleutelingsalgoritme zou ook onderzocht kunnen worden, maar aangezien het versleutelen van een bericht zeer snel is tegenover het genereren van een bewijs, zal dit een kleinere verbetering opleveren.

8.2 Partiële attestatie

In de attestatie implementatie van de SμV wordt het hele instructiegeheugen geattesteerd. In ons geval gebruikte de meest beveiligde applicatie maar de helft van het beschikbare instructiegeheugen. De andere helft die leeg is, wordt echter ook geattesteerd. Dit is nodig omdat een kwaadaardig programma zich zou kunnen verbergen in deze ruimte die leeg behoort te zijn. Een optimalisatie die ons mogelijk lijkt, is dat de SμV niet enkel zijn eigen stukje geheugen beschermt, maar al het geheugen dat niet gebruikt wordt door de applicatie. Als we dan uitgaan van een correcte werking van de SμV, moet een kleiner stuk van het instructiegeheugen geattesteerd worden. De bespaarde uitvoeringstijd is dan recht evenredig met de hoeveelheid niet gebruikt geheugen zoals in figuur 4 wordt geïllustreerd.

8.3 Probabilistische attestatie

Een andere mogelijkheid om een kleiner deel van het geheugen te attesteren bij ieder bericht is om een meer probabilistische aanpak te volgen. De prover kan iedere keer een willekeurig deel van het geheugen kiezen dat geattesteerd wordt. Bij deze policy krijgt de verifier natuurlijk wel een zwakkere zekerheid over de staat van de prover.

8.4 Kritieke berichten

Een laatste mogelijke optimalisatie die ons interessant lijkt is om vanuit de applicatie te beslissen welke berichten belangrijk genoeg zijn om een bewijs te genereren. Een mogelijke toepassing die we hier zijn is bv. Een apparaat dat naast metingen af en toe berichten uitwisselt om een klok te synchroniseren. Als deze klok niet essentieel is tot de veiligheid van het apparaat en/of de communicatie, zou het apparaat er voor kunnen opteren om geen integriteitsbewijs mee in het bericht te stoppen.

9 Conclusie

In dit onderzoek is aangetoond dat proof-carrying communicatie mogelijk is met een Arduino Uno rev 3. Deze Arduino voldoet aan de hardware vereisten om tot de klasse-0 IoT apparaten te behoren. Uit de waarnemingen is gebleken dat de overhead op het vlak van uitvoeringstijd en geheugengebruik niet te verwaarlozen is. Bij energiezuinige of tijdskritsiche applicaties kan de trade-off tussen extra beveiliging en overhead een lastige afweging zijn. Er zijn toepassingen waarbij het gebruik van proof-carrying messages te verantwoorden is. Apparaten met een lage frequentie van verstuurde berichten zullen minder bewijzen moeten opstellen wat resulteert in minder overhead. Voorbeelden hiervan zijn de IoT toestellen die verbonden zijn met de Sigfox en LoRa netwerken. Daarnaast zijn apparaten die een onbeperkte energiebron hebben zoals permanent geïnstalleerde brandalarmen mogelijke toepassing. Ook sensoren voor de verzameling van data kan een doelgroep vormen indien ze geen strikte tijdslimitaties hebben. Ten slotte kan Proof-carrying messages ook een bijdragen leveren aan IoT apparaten die beveiliging boven performantie stellen. Een steeds grotere groep valt binnen deze categorie omdat IoT toestellen steeds meer ingebed zitten in het dagelijks leven, waardoor beveiliging alsmaar belangrijker wordt vanwege privacy.

In dit onderzoek is er slechts een proof-of-concept geïmplementeerd. Er is nog vooruitgang te boeken op het gebied van performantie waardoor het aantal toepasbare situaties zal vergroten.

Referenties

[1] Ferran Adelantado e.a. „Understanding the limits of LoRaWAN". In: *IEEE Communications magazine* 55.9 (2017), p. 34–40.

[2] *Arduino Uno Rev3*. https://store.arduino.cc/arduino-uno-rev3. Arduino, 2020.

[3] Mihir Bellare. „New proofs for NMAC and HMAC: Security without collision-resistance". In: *Annual International Cryptology Conference*. Springer. 2006, p. 602–619.

[4] J. Christopher en Bare. *Attestation and Trusted Computing*. https : / / courses . cs . washington . edu / courses / csep590 / 06wi / finalprojects / bare . pdf. University of Washington, 2006.

[5] Wilfried Daniels e.a. „SμV - the Security Microvisor: A Virtualisation-Based Security Middleware for the Internet of Things". In: *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track*. Las Vegas, Nevada: Association for Computing Machinery, 2017, p. 36–42. ISBN: 9781450352000. DOI: 10 . 1145 / 3154448 . 3154454. URL: https://doi.org/10.1145/3154448.3154454.

[6] Aurélien Francillon e.a. „A minimalist approach to remote attestation". In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2014, p. 1–6.

[7] Ari Keranen, Mehmet Ersue en Carsten Bormann. *Terminology for Constrained-Node Networks*. RFC 4180. RFC Editor, apr 2014. URL: https://www.rfc-editor.org/rfc/rfc7228.txt.

[8] Chongkyung Kil e.a. „Remote attestation to dynamic system properties: Towards providing complete system integrity evidence". In: *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*. IEEE. 2009, p. 115–124.

[9] Vincent Rijmen en Elisabeth Oswald. *Update on SHA-1*. Cryptology ePrint Archive, Report 2005/010. https://eprint.iacr.org/2005/010. 2005.

[10] Rodrigo Vieira Steiner en Emil Lupu. „Attestation in wireless sensor networks: A survey". In: *ACM Computing Surveys (CSUR)* 49.3 (2016), p. 1–31.

[11] Dazhi Zhang en Donggang Liu. „DataGuard: Dynamic data attestation in wireless sensor networks". In: *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*. IEEE. 2010, p. 261–270.