Mécanismes d'authentification pour des réseaux de capteurs sans fil multi-sauts

Ismail Mansour, Pascal Lafourcade, Gérard Chalhoub

Mécanismes d'authentification pour des réseaux de capteurs sans fil multi-sauts †

Ismail Mansour, Pascal Lafourcade et Gérard Chalhoub

LIMOS, Clermont Université, Campus des Cézeaux, Aubière, France

Proposer des mécanismes d'authentification sécurisés pour associer de nouveaux nœuds dans les réseaux de capteurs sans fil (RCSF) n'est pas une tâche facile. Nous proposons plusieurs protocoles d'authentification multi-sauts pour les RCSF. Pour chacune de nos propositions nous prouvons formellement la sécurité de nos protocoles en utilisant l'outil automatique Scyther. Nous effectuons aussi une analyse du temps d'exécution sur des cartes TelosB. Nous comparons suivant ce critère les différents protocoles d'authentification mis en place.

Keywords: réseaux de capteurs, sécurité, authentification, multi-sauts, cartes TelosB, évaluation.

1 Introduction

Les réseaux de capteurs sans fil (RCSF) sont de plus en plus utilisés dans des applications critiques, où les participants doivent être authentifiés avant d'échanger des données. La nature sans fil de ces réseaux augmente les possibilités d'attaques d'une entité malicieuse. Sécuriser de tels réseaux nécessite souvent l'emploi de primitives cryptographiques coûteuses en temps de calcul. Une approche naïve consiste à concevoir des protocoles en utilisant le plus de primitives cryptographiques. Cela conduit souvent à un surcoût de consommation de ressources dans le réseau. De plus, l'utilisation de la cryptographie ne suffit pas à garantir la sécurité d'un protocole. Par exemple, le protocole d'authentification de nœuds (appelé JIS_{orig} dans la suite) proposé dans [4] souffre d'une attaque malgré l'utilisation d'algorithmes de chiffrement. Afin de corriger ce problème, nous proposons de nouveaux protocoles d'authentification de nœuds dans un RCSF multi-sauts. Tous nos protocoles ont été prouvés sûrs avec Scyther. Nous avons également programmé tous les protocoles étudiés sur des cartes TelosB afin de comparer leurs performances et de trouver le meilleur compromis entre sécurité et efficacité.

2 Protocoles d'authentification multi-sauts

Avant de présenter nos protocoles d'authentification, nous décrivons les primitives cryptographiques et les notations utilisées. Nous utilisons la cryptographie sur les courbes elliptiques (ECC), avec les paramètres secp160r1 et secp128r1 donnés par le « Standards for Efficient Cryptography Group » [5]. Notre implémentation d'ECC sur les TelosB est basée sur la bibliothèque TinyECC [3]. Plus précisément nous avons implémenté le protocole à clefs publiques ECIES (Elliptic Curve Integrated Encryption Scheme) inventé par Victor Shoup en 2001. Nous avons également utilisé un code optimisé pour le chiffrement symétrique AES [2].

Pour nos protocoles d'authentification nous avons proposé un mécanisme d'échange de clefs sans interaction en utilisant ECC. Cette approche est inspirée du protocole classique d'échange de clefs de Diffie-Hellman et fonctionne comme suit.

Avant le déploiement du réseau, chaque nœud N possède la clef publique $pk(S)$ du puits S ainsi que sa propre paire de clefs publique/privée ECC, notée $(pk(N), sk(N))$. De par l'utilisation d'ECC, la clef publique $pk(N)$ du nœud N vaut le produit de la clef privée $sk(N)$ et du point publique G générateur de la courbe choisie. Avec cette connaissance chaque nœud peut calculer une clef partagée $K_{DH}(N,S)$ avec le

†Ces travaux ont été réalisés avec le soutien de la Chaire « Confiance numérique » de la Fondation de l'Université d'Auvergne.

Protocole	Authentification	Clef	de R à S		de S à R	
			chiffrement	Déchiffrement	Chiffrement	Déchiffrement
JIS_{orig}	non	$K_{DH}(N,S)$	non	non	non	non
JIS_{NK}	oui	NK	oui	oui	oui	oui
JIS_{K}	oui	$K(J_i,J_{i+1})$	oui	oui	oui	oui
$JIS_{NK,onion}$	oui	NK	oui	non	non	oui

TABLE 1: Opérations faites par les nœuds intermédiaires.

puits S. Ces calculs coûteux peuvent être faits par le puits pour tous les nœuds avant la phase de déploiement afin d'économiser de l'énergie. Le puits connaît sa propre clef secrète $sk(S)$ et la clef publique $pk(N)$ du nœud N. Il calcule alors $K_{DH}(N,S) = sk(S) \times pk(N)$. De son côté le nœud N multiplie sa clef $sk(N)$ avec la clef publique du puits $pk(S)$ afin d'obtenir $K_{DH}(N,S)$. Les deux calculs donnent la même clef partagée car :

$$K_{DH}(N,S) = sk(N) \times pk(S) = sk(N) \times (sk(S) \times G) = (sk(N) \times G) \times sk(S) = pk(N) \times sk(S)$$

Dans la suite, nous utilisons les notations suivantes : I représente le nœud qui initie le protocole d'authentification ; R est un voisin du nœud I ; S est le nœud puits du réseau ; J_i est le i-ème nœud intermédiaire entre R et S choisi par le protocole de routage ; n_A dénote un nonce (nombre aléatoire frais) généré par le nœud A ; $pk(A)$ est la clef publique du nœud A ; $sk(A)$ la clef privée du nœud A ; $K(I,S)$ la clef de session entre les nœuds I et S ; NK est la clef symétrique partagée entre tous les nœuds du réseau ; $K_{DH}(N,S)$ dénote la clef symétrique entre les nœuds N et S construite en utilisant le mécanisme d'échange de clefs sans interaction ; $\{x\}_k$ est le chiffrement du message x avec la clef symétrique ou asymétrique k.

2.1 Joindre Directement S : JDS_{orig}

Le protocole JDS_{orig} presenté dans [4], permet à un nouveau nœud I dans le voisinage de S de rejoindre directement le réseau. Pour cela I calcule la clef symétrique $K_{DH}(I,S)$, génére un nonce n_I et envoie $\{n_I,I\}_{K_{DH}(I,S)}$ à S. Ensuite S calcule $K_{DH}(I,S)$ et déchiffre le message reçu. Puis il vérifie l'identité de I et génére une nouvelle clef symétrique de session $K(I,S)$. Il répond alors à I avec $\{n_I,S,K(I,S)\}_{pk(I)}$. Seul I peut déchiffrer ce message avec $sk(I)$. Remarquons que le nonce n_I sert à l'authentification des réponses.

2.2 Joindre Indirectement S

Nous proposons plusieurs protocoles pour qu'un nœud hors de portée de S puisse rejoindre le réseau via des nœuds déjà authentifiés. Chacune de nos solutions utilise un mécanisme d'authentification différent, la Table 1 montre les différences entre chaque protocole.

JIS_{orig} : Dans ce protocole présenté dans [4], le nœud I génére un nonce n_I et envoie $\{n_I,I\}_{K_{DH}(I,S)}$ à R un de ses voisins authentifiés dans le réseau. Ce message est transféré jusqu'à atteindre S. Le puits renvoie par le même chemin le message $\{n_I,S,pk(I)\}_{K_{DH}(R,S)}$ à R. Enfin, R renvoie à I le message $\{n_I,R,K(R,I)\}_{pk(I)}$ contenant la clé de session $K(R,I)$ après l'avoir créée accompagnée du nonce n_I et l'identité de R.

Ce protocole suppose les nœuds intermédiaires de confiance. Ainsi comme nous l'avons constaté avec Scyther, il n'est pas résistant à une attaque d'authentification en présence d'un nœud intermédiaire est compromis. Dans la suite nous proposons des protocoles qui permettent de joindre indirectement le réseau sans supposer que les nœuds intermédiaires soient de confiance et ceci en authentifiant chacun de ces noeuds à chaque saut. Par conséquence, le chemin suivi par le message au retour correspond à celui suivi à l'aller et les liens entre les noeuds sont donc bidirectionnels.

JIS_{NK} : L'idée de ce protocole est d'ajouter un nonce à chaque saut afin d'authentifier tous les participants. Cette méthode nécessite un déchiffrement et un chiffrement de la part des nœuds intermédiaires. Le nœud I envoie $\{n_I,I,R\}_{K_{DH}(I,S)}$ à son voisin R qui génére un nonce n_R l'ajoute au message reçu et le chiffre avec NK avant de l'envoyer à J. Ensuite J déchiffre le message avec NK, ajoute un nonce n_J et chiffre le tout avec NK avant de l'envoyer au suivant. Le puits déchiffre le message, récupère tous les nonces et construit le message suivant $\{pk(I),\{n_I,I,R,S\}_{K_{DH}(I,S)}\}_{K_{DH}(R,S)}$ auquel il ajoute les nonces de tous les nœuds intermédiaires. Il

renvoie donc le tout chiffré avec NK. Chaque nœud intermédiaire déchiffre et vérife la présence de son nonce, avant de renvoyer le message chiffré sans son nonce au suivant. Enfin R crée une clef symétrique de session $K(R,I)$ et envoie à I le message $\{K(R,I), \{n_I, I, R, S\}_{K_{DH}(I,S)}\}_{pk(I)}$.

JIS_K : Ce protocole fonctionne comme JIS_{NK} mais au lieu d'utiliser la clef NK chaque nœud utilise sa clef symétrique $K(J_i, J_{i+1})$ partagée avec le nœud intermédiaire suivant. De plus I ajoute l'ensemble des nœuds inermédiaires dans son premier message ; pour ce faire, nous supposons que I peut obtenir un chemin sûr entre R et S. Remarquons que R connaît déjà un tel chemin car il a pu rejoindre le réseau par cette même méthode.

$JIS_{NK,onion}$: Afin d'éviter aux nœuds intermédiaires de déchiffrer puis chiffrer chaque message, ils ne vont pas déchiffrer mais uniquement ajouter leur nonce et chiffrer le message reçu. Ce protocole est donc une amélioration en consommation énergétique du protocole JIS_{NK}. Ainsi, les nœuds intermédiaires construissent un "oignon" que S va peler, tout en gardant les nonces des nœuds intermédiaires et afin de modifier le message au centre de ces chiffrements. Ensuite il reconstruit l'oignon avec ce nouveau message. Chaque nœud intermédiaire va donc enlever un seul chiffrement dans la phase retour. Ce protocole diminue le nombre d'opérations faites par les nœuds intermédiaires et augmente le travail du puits que nous supposons disposer de plus de ressources que les autres nœuds du réseau.

Nous avons utilisé Scyther [1], un des outils de vérification de protocoles cryptographiques les plus rapides et les plus conviviaux à utiliser, pour trouver la faille d'authentification sur le protocole original et pour prouver la sécurité de tous les autres protocoles.

3 Évaluation des performances des protocoles

La principale différence entre nos protocoles vient de l'usage des primitives cryptographiques. Nous analysons les deux scénarios suivants : 1) nous utilisons CTR AES avec une clef de 128 bits et ECIES avec une clef de 128 bits ; 2) cette fois-ci nous utilisons une clef de 160 bits pour ECIES. Dans la table 2, nous donnons les temps d'exécution de notre implémentation sur des cartes TelosB pour 4 nœuds : I, R, J, et S[‡]. De plus, nous avons simulé le temps d'exécution de chaque protocole avec notre propre simulateur écrit en C, qui additionne les coûts des primitives utilisées ; ceci afin d'observer le comportement de nos protocoles pour plus de nœuds. La seconde colonne de la table montre que les résultats simulés sont cohérents avec les mesures réalisées.

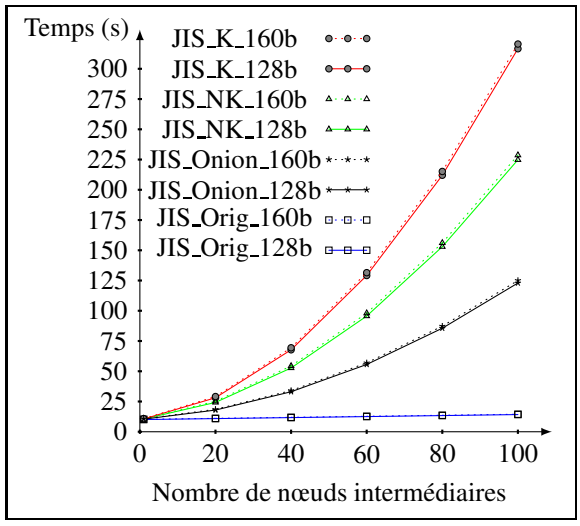

FIGURE 1: Temps d'exécution de la phase d'authentification d'un nouveau nœud.

Nous considérons que le puits a plus de ressources de calcul que les autres nœuds ainsi nous ne comptons pas les calculs faits par S. Dans la Figure 1, nous présentons le temps d'exécution pour tous nos protocoles pour les deux scénarios pour 20, 40, 60, 80 et 100 nœuds intermédiaires. Nous remarquons d'abord que la taille de la clef ne change pas les résultats. Ceci est dû au fait que le code utilisé est optimisé pour une clef de 160 bits. En faisant croître le nombre de nœuds intermédiaires nous voyons clairement l'impact du coût des primitives cryptographiques. Soulignons que le nombre de nœuds intermédiaires correspond approximativement à la moitié du diamètre du réseau et non au nombre de nœuds du réseau. Nous observons que le temps d'exécution du protocole original est quasi-invariant quand le nombre de nœuds intermédiaires augmente, ceci car ces nœuds ne servent que de relais dans ce protocole et les calculs sont faits par I et S. Ensuite le protocole JIS_{onion}

‡. Ces mesures sont la moyenne de 20 expériences pour chaque scénario.

scenario 1 - ECIES - 128b			
Protocole	Temps d'exécution réel [ms]	Temps d'exécution simulé [ms]	Taille des messages [octets]
JDS	3791.20	3764.00	62
JIS_{orig}	10032.20	10079.20	146
JIS_{NK}	10492.45	10538.16	179
JIS_{K}	10492.32	10538.88	185
$JIS_{NK,onion}$	10376.05	10398.16	179
scenario 2 - ECIES - 160b			
Protocole	Temps d'exécution réel [ms]	Temps d'exécution simulé [ms]	Taille des messages [octets]
JDS	4093.60	4116.48	66
JIS_{orig}	10169.95	10268.60	166
JIS_{NK}	10804.21	10832.56	199
JIS_{K}	10790.95	10833.28	205
$JIS_{NK,onion}$	10636.11	10657.56	199

TABLE 2: Le temps de rejoindre le réseau à travers un nœud intermédiaire avec la taille de tous les messages échangés.

est plus efficace que les protocoles JIS_{NK} et JIS_{K} car le nombre global d'opérations crypotgraphiques est moindre dans les nœuds intermédiaires. Enfin le protocole JIS_{NK} est le plus lent, car le volume de données à chiffrer est plus important.

4 Conclusion et discussion

Nous avons proposé plusieurs protocoles d'authentification multi-sauts pour les RCSF. En utilisant l'outil Scyther, nous avons prouvé la sécurité de nos protocoles. De plus, nous avons implémenté tous nos protocoles sur des cartes TelosB afin d'analyser leurs performances. Le premier protocole suppose que les messages ne suivent pas les mêmes chemins à l'aller et au retour, par contre il faut faire confiance à l'ensemble des nœuds intermédiaires. Ceci réduit considérablement le nombre d'opérations cryptographiques à faire. Cette solution passe à l'échelle pour des réseaux comportant des milliers de nœuds.

D'autre part, de nombreuses applications nécessitent non seulement une authentification de bout en bout mais aussi une authentification par saut. Pour cela nous avons proposé 3 protocoles et évalué leurs performances. Ces protocoles diffèrent de par leur temps d'exécution. Pour une ligne contenant 100 nœuds intermédiaires, l'authentification par saut a un surcoût non négligeable qui est pour la solution la plus lente de 5 minutes et 20 secondes et de 2 minutes pour la plus rapide.

Nous continuons ces travaux en regardant les mécanismes de révocation de clefs pour les RCSF. Ces mécanismes sont cruciaux pour assurer le bon fonctionnement d'un réseau où de nombreux nœuds apparaissent et disparaissent. Le principal défi est de proposer des protocoles sûrs et peu gourmands en ressources de calcul afin de pouvoir les déployer sur des nœuds à faible capacité de calcul.

Références

[1] C. Cremers. The Scyther Tool : Verification, falsification, and analysis of security protocols. In *Computer Aided Verification, 20th International Conference, CAV 2008, Proc.*, volume 5123/2008, 2008.

[2] J. Daemen and V. Rijmen. *The Design of Rijndael : AES - The Advanced Encryption Standard*. 2002.

[3] A. Liu and N. Ning. Tinyecc : A configurable library for elliptic curve cryptography in wireless sensor networks. In *7th International Conference on Information Processing in Sensor Networks*, 2008.

[4] I. Mansour, G. Chalhoub, and M. Misson. *Security architecture for multi-hop wireless sensor networks*. CRC Press Book, 2014.

[5] C. Research. Standards for efficient cryptography, sec 1 : Elliptic curve cryptography, September 2000.