

Internet Engineering Task Force	D. M'Raihi
Internet-Draft	Verisign, Inc.
Intended status: Informational	S. Machani
Expires: August 29, 2011	Diversinet Corp.
	M. Pei
	Symantec
	J. Rydell
	Portwise, Inc.
	February 25, 2011

TOTP: Time-based One-time Password Algorithm  
draft-mraihi-totp-timebased-08.txt

## Abstract

This document describes an extension of one-time password (OTP) algorithm, namely the HMAC-Based One-Time Password (HOTP) Algorithm as defined in RFC 4226, to support time-based moving factor. The HOTP algorithm specifies an event based OTP algorithm where the moving factor is an event counter. The present work bases the moving factor on a time value. A time-based variant of the OTP algorithm provides short-lived OTP values, which are desirable for enhanced security. The proposed algorithm can be used across a wide range of network applications ranging from remote Virtual Private Network (VPN) access, Wi-Fi network logon to transaction-oriented Web applications. The authors believe that a common and shared algorithm will facilitate adoption of two-factor authentication on the Internet by enabling interoperability across commercial and open-source implementations.

## Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 29, 2011.

## Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license->

info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## [Table of Contents](#)

- \*1. [Introduction](#)
- \*1.1. [Scope](#)
- \*1.2. [Background](#)
- \*2. [Notation and Terminology](#)
- \*3. [Algorithm Requirements](#)
- \*4. [TOTP Algorithm](#)
- \*4.1. [Notations](#)
- \*4.2. [Description](#)
- \*5. [Security Considerations](#)
- \*5.1. [General](#)
- \*5.2. [Validation and Time-step Size](#)
- \*6. [Resynchronization](#)
- \*7. [IANA Considerations](#)
- \*8. [Acknowledgements](#)
- \*9. [References](#)
- \*9.1. [Normative references](#)
- \*9.2. [Informative References](#)
- \*Appendix A. [TOTP Algorithm: Reference Implementation](#)
- \*Appendix B. [Test Vectors](#)
- \*[Authors' Addresses](#)

## **1. Introduction**

### **1.1. Scope**

This document describes an extension of one-time password (OTP) algorithm HMAC-Based One-Time Password (HOTP) as defined in [\[RFC4226\]](#) to support time based moving factor.

### **1.2. Background**

As defined in [\[RFC4226\]](#) the HOTP algorithm is based on the HMAC-SHA-1 algorithm, as specified in [\[RFC2104\]](#) applied to an increasing counter value representing the message in the HMAC computation.

Basically, the output of the HMAC-SHA-1 calculation is truncated to obtain user-friendly values:

$$\text{HOTP}(K,C) = \text{Truncate}(\text{HMAC-SHA-1}(K,C))$$

where Truncate represents the function that can convert an HMAC-SHA-1 value into an HOTP value. K and C represent the shared secret and counter value, see [\[RFC4226\]](#) for their detail definition.

TOTP is the time-based variant of this algorithm where a value T derived from a time reference and a time step replaces the counter C in the HOTP computation.

TOTP implementations MAY use HMAC-SHA-256 or HMAC-SHA-512 functions, based on SHA-256 or SHA-512 [SHA2] hash functions, instead of HMAC-SHA-1 function that has been specified for HOTP computation in [\[RFC4226\]](#).

## **2. Notation and Terminology**

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [\[RFC2119\]](#).

## **3. Algorithm Requirements**

This section summarizes the requirements taken into account for designing the TOTP algorithm.

R1 - The prover (e.g. token, soft token) and verifier (authentication or validation server) MUST know or be able to derive the current Unix time (i.e. the number of seconds elapsed since midnight UTC of January 1, 1970) for OTP generation. See [UT] for more detail definition of the commonly known "Unix time". The precision of the time used by the prover affects how often the clock synchronization should be done, see Section 6.

R2 - The prover and verifier MUST either share a same secret or the knowledge of a secret transformation to generate a shared secret

R3 - The algorithm MUST use HOTP [\[RFC4226\]](#) as a key building block.

R4 - The prover and verifier MUST use the same time step value X.

R5 - There MUST be a unique secret (key) for each prover.

R6 - The keys SHOULD be randomly generated or derived using a key derivation algorithms.

R7 - The keys MAY be stored in a tamper-resistant device and SHOULD be protected against unauthorized access and usage.

#### [4. TOTP Algorithm](#)

This variant of the HOTP algorithm specifies the calculation of a one-time password value, based on a representation of the counter as a time factor.

##### [4.1. Notations](#)

- X represents the time step in seconds (default value X = 30 seconds) and is a system parameter;
- T0 is the Unix time to start counting time steps (default value is 0, Unix epoch) and is also a system parameter.

##### [4.2. Description](#)

Basically, we define TOTP as  $TOTP = HOTP(K, T)$  where T is an integer and represents the number of time steps between the initial counter time T0 and the current Unix time.

More specifically  $T = (Current\ Unix\ time - T0) / X$  where:

- X represents the time step in seconds (default value X = 30 seconds) and is a system parameter.
- T0 is the Unix time to start counting time steps (default value is 0, Unix epoch) and is also a system parameter;
- The default floor function is used in the computation. For example, with  $T0 = 0$  and time step  $X = 30$ ,  $T = 1$  if the current Unix time is 59 seconds and  $T = 2$  if the current Unix time is 60 seconds.

The implementation of this algorithm MUST support the time value T larger than 32-bit integer when it is beyond year 2038. The value of the system parameters X and T0 are pre-established during the provisioning process and communicated between a prover and verifier as part of the provisioning step. The provisioning flow is out of scope of this document, refer to [RFC6030] for such provisioning container specification.

#### [5. Security Considerations](#)

##### [5.1. General](#)

The security and strength of this algorithm depends on the properties of the underlying building block HOTP, which is a construction based on HMAC [\[RFC2104\]](#) using SHA-1 as the hash function.

The conclusion of the security analysis detailed in [\[RFC4226\]](#) is that, for all practical purposes, the outputs of the dynamic truncation on distinct inputs are uniformly and independently distributed strings.

The analysis demonstrates that the best possible attack against the HOTP function is the brute force attack.

As indicated in the algorithm requirement section, keys SHOULD be chosen at random or using a cryptographically strong pseudo-random generator properly seeded with a random value.

Keys SHOULD be of the length of the HMAC output to facilitate interoperability.

We RECOMMEND following the recommendations in [\[RFC4086\]](#) for all pseudo-random and random generations. The pseudo-random numbers used for generating the keys SHOULD successfully pass the randomness test specified in [\[CN\]](#) or a similar well-recognized test.

All the communications SHOULD take place over a secure channel e.g. SSL/TLS [\[RFC5246\]](#), IPsec connections [\[RFC4301\]](#).

We also RECOMMEND storing the keys securely in the validation system, and more specifically encrypting them using tamper-resistant hardware encryption and exposing them only when required: for example, the key is decrypted when needed to verify an OTP value, and re-encrypted immediately to limit exposure in the RAM for a short period of time.

The key store MUST be in a secure area, to avoid as much as possible direct attack on the validation system and secrets database.

Particularly, access to the key material should be limited to programs and processes required by the validation system only.

## **[5.2. Validation and Time-step Size](#)**

An OTP generated within the same Time-step will be the same. When an OTP is received at a validation system, it doesn't know a client's exact timestamp when an OTP was generated. The validation system may typically use the timestamp when an OTP is received for OTP comparison. Due to the network latency for an OTP to transmit from a requesting application to a validation system and user's actual input time of an OTP to a receiving system, such timestamp gap between the actual OTP generation time and server's receiving time may be large. The receiving time at the validation system and the actual OTP generation may not fall within the same Time-step window that produce the same OTP. When an OTP is generated at the end of a Time-step window, the receiving time most likely falls into the next Time-step window. A validation system SHOULD typically set a policy for an acceptable OTP transmission delay window for validation. The validation system should compare OTPs not only with the receiving timestamp but also the past timesteps that are within the transmission delay. A larger acceptable delay window would introduce some OTP attack window. We RECOMMEND that at most one time step is allowed as the network delay.

The Time-step size has impact on both security and usability. A larger Time-step size means larger validity window for an OTP to be accepted by a validation system. There are the following implications with a larger Time-step size.

At first, a larger Time-step size exposes larger window for attack. When an OTP is generated and exposed to a third party before it is

consumed, the third party can consume the OTP within the Time-step window.

We RECOMMEND default Time-step size for 30 seconds. This default value of 30 seconds is selected to balance between security and usability. Secondly, the next different OTP must be generated in the next Time-step window. A user must wait till the clock moves to the next Time-step window from the last submission. The waiting time may not be exactly the length of Time-step depending on when the last OTP was generated. For example, if the last OTP was generated at the half way in a Time-step window, the waiting time for the next OTP is half of length of Time-step. In general, a larger Time-step window means larger waiting time for a user to get the next valid OTP after the last successfully OTP validation. A too large window, for example 10 minutes, most probably won't be suitable for typical internet login use cases; a user may not be able to get the next OTP within 10 minutes and therefore re-login back to the same site in 10 minutes.

Note that a prover may send the same OTP inside a given time window multiple times to a verifier. The verifier MUST not accept the second attempt of the OTP after the successful validation has been issued for the first OTP, which ensures one-time only use of an OTP.

## **6. Resynchronization**

Because of possible clock drifts between a client and a validation server, we RECOMMEND that the validator be set with a specific limit to the number of time steps a prover can be 'out of synch' before being rejected.

This limit can be set both forward and backwards from the calculated time step on receipt of the OTP value. If the time step is 30 seconds as recommended, and the validator is set to only accept 2 time step backwards then the maximum elapsed time drift would be around 89 seconds, i.e. 29 seconds in the calculated time step and 60 for two backward time steps.

This would mean the validator could perform a validation against the current time and then further two validations for each backward step (for a total of 3 validations). Upon successful validation, the validation server can record the detected clock drift for the token in terms of number of Time-step. When a new OTP is received after this step, the validator can validate the OTP with current timestamp adjusted with recorded number of Time-step clock drifts for the token. Also, it is important to note that the longer a prover has not sent an OTP to a validation system, the longer (potentially) the accumulated clock drift between the prover and the verifier. In such cases, the automatic resynchronization described above may not work if the drift exceeds the allowed threshold. Additional authentication measures should be used to safely authenticate the prover and explicitly resynchronize the clock drift between the prover and the validator.

## [7. IANA Considerations](#)

This document has no actions for IANA.

## [8. Acknowledgements](#)

The authors of this draft would like to thank the following people for their contributions and support to make this a better specification: Hannes Tschofenig, Jonathan Tulliani, David Dix, Siddharth Bajaj, Stu Veath, Shuh Chang, Oanh Hoang, John Huang, and Siddhartha Mohapatra.

## [9. References](#)

### [9.1. Normative references](#)

, "

[RFC4226]	M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D. and O. Ranen, " <a href="#">HOTP: An HMAC-Based One-Time Password Algorithm</a> ", RFC 4226, December 2005.
[RFC4086]	Eastlake, D., Schiller, J. and S. Crocker, " <a href="#">Randomness Recommendations for Security</a> ", RFC 4086, June 2005.
[RFC2119]	<a href="#">Key words for use in RFCs to Indicate Requirement Levels</a> ", BCP 14, RFC 2119, March 1997.
[RFC2104]	Krawczyk, H., Bellare, M. and R. Canetti, " <a href="#">HMAC: Keyed-Hashing for Message Authentication</a> ", RFC 2104, February 1997.
[SHA2]	NIST, , "FIPS PUB 180-3: Secure Hash Standard (SHS)", October 2008.

### [9.2. Informative References](#)

[CN]	Coron, J.S. and D. Naccache, "An accurate evaluation of Maurer's universal test", LNCS 1556, February 1999.
[RFC4301]	Kent, S. and K. Seo, " <a href="#">Security Architecture for the Internet Protocol</a> ", RFC 4301, December 2005.
[RFC5246]	Dierks, T. and E. Rescorla, " <a href="#">The Transport Layer Security (TLS) Protocol Version 1.2</a> ", RFC 5246, August 2008.
[RFC6030]	Hoyer, P., Pei, M. and S. Machani, " <a href="#">Portable Symmetric Key Container (PSKC)</a> ", RFC 6030, October 2010.
[UT]	, , "Unix time", February 2011.

## [Appendix A. TOTP Algorithm: Reference Implementation](#)

<CODE BEGINS>

/\*\*

Copyright (c) 2011 IETF Trust and the persons identified as authors of the code. All rights reserved.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*/

```
import java.lang.reflect.UndeclaredThrowableException;
import java.security.GeneralSecurityException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.math.BigInteger;
import java.util.TimeZone;
```

/\*\*

\* This an example implementation of the OATH TOTP algorithm.

\* Visit [www.openauthentication.org](http://www.openauthentication.org) for more information.

\*

\* @author Johan Rydell, PortWise, Inc.

\*/

```
public class TOTP {
```

```
    private TOTP() {}
```

/\*\*

\* This method uses the JCE to provide the crypto

\* algorithm.

\* HMAC computes a Hashed Message Authentication Code with the

\* crypto hash algorithm as a parameter.

\*

\* @param crypto the crypto algorithm (HmacSHA1, HmacSHA256,

\* HmacSHA512)

\* @param keyBytes the bytes to use for the HMAC key

\* @param text the message or text to be authenticated.



```

    */
private static byte[] hmac_sha(String crypto, byte[] keyBytes,
    byte[] text){
    try {
        Mac hmac;
        hmac = Mac.getInstance(crypto);
        SecretKeySpec macKey =
            new SecretKeySpec(keyBytes, "RAW");
        hmac.init(macKey);
        return hmac.doFinal(text);
    } catch (GeneralSecurityException gse) {
        throw new UndeclaredThrowableException(gse);
    }
}

/**
 * This method converts HEX string to Byte[]
 *
 * @param hex the HEX string
 *
 * @return A byte array
 */
private static byte[] hexStr2Bytes(String hex){
    // Adding one byte to get the right conversion
    // values starting with "0" can be converted
    byte[] bArray = new BigInteger("10" + hex,16).toByteArray();

    // Copy all the REAL bytes, not the "first"
    byte[] ret = new byte[bArray.length - 1];
    for (int i = 0; i < ret.length ; i++)
        ret[i] = bArray[i+1];
    return ret;
}

private static final int[] DIGITS_POWER
// 0 1 2 3 4 5 6 7 8
= {1,10,100,1000,10000,100000,1000000,10000000,100000000 };

/**
 * This method generates an TOTP value for the given
 * set of parameters.
 *
 * @param key the shared secret, HEX encoded
 * @param time a value that reflects a time
 * @param returnDigits number of digits to return
 *
 * @return A numeric String in base 10 that includes

```

```

        *           {@link truncationDigits} digits
    */
    public static String generateTOTP(String key,
        String time,
        String returnDigits){
        return generateTOTP(key, time, returnDigits, "HmacSHA1");
    }

    /**
     * This method generates an TOTP value for the given
     * set of parameters.
     *
     * @param key the shared secret, HEX encoded
     * @param time a value that reflects a time
     * @param returnDigits number of digits to return
     *
     * @return A numeric String in base 10 that includes
     *         {@link truncationDigits} digits
     */
    public static String generateTOTP256(String key,
        String time,
        String returnDigits){
        return generateTOTP(key, time, returnDigits, "HmacSHA256");
    }

    /**
     * This method generates an TOTP value for the given
     * set of parameters.
     *
     * @param key the shared secret, HEX encoded
     * @param time a value that reflects a time
     * @param returnDigits number of digits to return
     *
     * @return A numeric String in base 10 that includes
     *         {@link truncationDigits} digits
     */
    public static String generateTOTP512(String key,
        String time,
        String returnDigits){
        return generateTOTP(key, time, returnDigits, "HmacSHA512");
    }

    /**
     * This method generates an TOTP value for the given
     * set of parameters.
     *

```

```

* @param key the shared secret, HEX encoded
* @param time a value that reflects a time
* @param returnDigits number of digits to return
* @param crypto the crypto function to use
*
* @return A numeric String in base 10 that includes
*         {@link truncationDigits} digits
*/
public static String generateTOTP(String key,
    String time,
    String returnDigits,
    String crypto){
    int codeDigits = Integer.decode(returnDigits).intValue();
    String result = null;

    // Using the counter
    // First 8 bytes are for the movingFactor
    // Complaint with base RFC 4226 (HOTP)
    while(time.length() < 16 )
        time = "0" + time;

    // Get the HEX in a Byte[]
    byte[] msg = hexStr2Bytes(time);
    byte[] k = hexStr2Bytes(key);

    byte[] hash = hmac_sha(crypto, k, msg);

    // put selected bytes into result int
    int offset = hash[hash.length - 1] & 0xf;

    int binary =
        ((hash[offset] & 0x7f) << 24) |
        ((hash[offset + 1] & 0xff) << 16) |
        ((hash[offset + 2] & 0xff) << 8) |
        (hash[offset + 3] & 0xff);

    int otp = binary % DIGITS_POWER[codeDigits];

    result = Integer.toString(otp);
    while (result.length() < codeDigits) {
        result = "0" + result;
    }
    return result;
}

public static void main(String[] args) {
    // Seed for HMAC-SHA1 - 20 bytes
    String seed = "3132333435363738393031323334353637383930";
    // Seed for HMAC-SHA256 - 32 bytes

```

```
String seed32 = "3132333435363738393031323334353637383930" +
"313233343536373839303132";
// Seed for HMAC-SHA512 - 64 bytes
String seed64 = "3132333435363738393031323334353637383930" +
"3132333435363738393031323334353637383930" +
"3132333435363738393031323334353637383930" +
"31323334";
long T0 = 0;
long X = 30;
long testTime[] = {59L, 1111111109L, 1111111111L,
1234567890L, 2000000000L, 2000000000L};

String steps = "0";
DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
df.setTimeZone(TimeZone.getTimeZone("UTC"));

try{
    System.out.println(
        "+-----+-----+" +
        "-----+-----+-----+");
    System.out.println(
        "| Time(sec) | Time (UTC format) | " +
        "| Value of T(Hex) | TOTP | Mode |");
    System.out.println(
        "+-----+-----+" +
        "-----+-----+-----+");

    for(int i=0; i<testTime.length; i++) {
        long T = (testTime[i] - T0)/X;
        steps = Long.toHexString(T).toUpperCase();
        while(steps.length() < 16) steps = "0" + steps;
        String fmtTime = String.format("%1$-11s", testTime[i]);
        String utcTime = df.format(new Date(testTime[i]*1000));
        System.out.print("| " + fmtTime + " | " + utcTime +
            " | " + steps + " |");
        System.out.println(generateTOTP(seed, steps, "8",
            "HmacSHA1") + "| SHA1 |");
        System.out.print("| " + fmtTime + " | " + utcTime +
            " | " + steps + " |");
        System.out.println(generateTOTP(seed32, steps, "8",
            "HmacSHA256") + "| SHA256 |");
        System.out.print("| " + fmtTime + " | " + utcTime +
            " | " + steps + " |");
        System.out.println(generateTOTP(seed64, steps, "8",
            "HmacSHA512") + "| SHA512 |");

        System.out.println(
            "+-----+-----+" +
            "-----+-----+-----+");
    }
}
```

```

    }
} catch (final Exception e){
    System.out.println("Error : " + e);
}
}
}

```

<CODE ENDS>

## Appendix B. Test Vectors

This section provides test values that can be used for HOTP time-based variant algorithm interoperability test.

The test token shared secret uses the ASCII string value "12345678901234567890". With Time Step  $X = 30$ , and Unix epoch as initial value to count time steps where  $T_0 = 0$ , the TOTP algorithm will display the following values for specified modes and timestamps.

Time (sec)	UTC Time	Value of T (hex)	TOTP	Mode
59	1970-01-01 00:00:59	000000000000000001	94287082	SHA1
59	1970-01-01 00:00:59	000000000000000001	46119246	SHA256
59	1970-01-01 00:00:59	000000000000000001	90693936	SHA512
1111111109	2005-03-18 01:58:29	00000000023523EC	07081804	SHA1
1111111109	2005-03-18 01:58:29	00000000023523EC	68084774	SHA256
1111111109	2005-03-18 01:58:29	00000000023523EC	25091201	SHA512
1111111111	2005-03-18 01:58:31	00000000023523ED	14050471	SHA1
1111111111	2005-03-18 01:58:31	00000000023523ED	67062674	SHA256
1111111111	2005-03-18 01:58:31	00000000023523ED	99943326	SHA512
1234567890	2009-02-13 23:31:30	000000000273EF07	89005924	SHA1
1234567890	2009-02-13 23:31:30	000000000273EF07	91819424	SHA256
1234567890	2009-02-13 23:31:30	000000000273EF07	93441116	SHA512
2000000000	2033-05-18 03:33:20	0000000003F940AA	69279037	SHA1
2000000000	2033-05-18 03:33:20	0000000003F940AA	90698825	SHA256
2000000000	2033-05-18 03:33:20	0000000003F940AA	38618901	SHA512
2000000000	2603-10-11 11:33:20	0000000027BC86AA	65353130	SHA1
2000000000	2603-10-11 11:33:20	0000000027BC86AA	77737706	SHA256
2000000000	2603-10-11 11:33:20	0000000027BC86AA	47863826	SHA512

TOTP Table

### Authors' Addresses

David M'Raihi M'Raihi Verisign, Inc. 685 E. Middlefield Road  
Mountain View, CA 94043 USA EMail: [davidietf@gmail.com](mailto:davidietf@gmail.com)

Salah Machani Machani Diversinet Corp. 2225 Sheppard Avenue East,  
Suite 1801 Toronto, Ontario M2J 5C2 Canada EMail:  
[smachani@diversinet.com](mailto:smachani@diversinet.com)

Mingliang Pei Pei Symantec 510 E. Middlefield Road  
Mountain View, CA 94043 USA EMail: [Mingliang\\_Pei@symantec.com](mailto:Mingliang_Pei@symantec.com)

Johan Rydell Rydell Portwise, Inc. 275 Hawthorne Ave, Suite 119 Palo  
Alto, CA 94301 USA EMail: [johan.rydell@portwise.com](mailto:johan.rydell@portwise.com)